conference

proceedings

Workshop on Embedded Systems

Cambridge, Massachusetts
March 29–31, 1999

Co-Sponsored by
The USENIX Association and
The MIT Media Laboratory

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

USENIX Proceedings of the Workshop on Embedded Systems

Cambridge, Massachusetts, USA, March 1999

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
WWW URL: http://www.usenix.org

The price is $20 for members and $28 for nonmembers.
Outside the U.S.A. and Canada, please add
$12 per copy for postage (via air printed matter).

ISBN 1-880446-31-6

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the Workshop on Embedded Systems

March 29–31, 1999
Cambridge, Massachusetts, USA

Workshop Organizers

Workshop Co-Chairs

Dan Geer, *CertCo*
Mike Hawley, *MIT Media Lab*

Program Committee

Mark D. Baushke, *Cisco Systems*
Warren Bosch, *Hasbro*
Tom Kalil, *The White House*
Tim Matt, *Siebe*
Jean Scholtz, *DARPA*
Ted Selker, *IBM*
Randy Sweeney, *Kraft Foods*
Jim Waldo, *Sun Microsystems*
Kevin Kelly, *WIRED magazine*

The USENIX Association Staff

Contents

Message from the Workshop Co-Chairs

Many of us grew up with the pre-PC and PC industries: as kids, we tinkered with computers using punched cards, paper tape, toggling hex-coded instructions onto cpu control panels, or playing Star Trek games that pecked out galactic maps on Diablo teletype terminals. Grownups back then were just beginning to invent pixels and mice for pushing pictures around. Today, vast industries have welled up around the paradigm of interacting through graphical interfaces. Forums like SIGGRAPH, once just a splinter group, are now supercharged by publishing, internet, and entertainment industries. For most people, the look and feel of computers is now synonymous with visual interface terminals. But these terminals are not the end of the line. In fact, they are barely the beginning.

Real humans interact through a plurality of richer modes: through touch and gesture, taste and smell, in noisy environments that are spatially, aurally, visually, physically, and emotionally intense. The world is our interface to the universe. Computer-mediated interfaces are destined to illuminate much of that. Digital technologies (computing, communication, memories, sensory links) will likely dissolve into every nook, cranny, and artifact in our environs. Future computers will surely be wearable, edible, implantable, and more.

It is just starting to happen. Kids growing up today, playing with toys like Hasbro's "Furby," hold in their hands a plush toy with more than four times the processing power of the Apollo lunar landing module. And it's barely a peashooter. In just a few years, when dolls and toys are invisibly linked to the net and infinitely fueled by it, older kids will sneer at how crude and utterly disconnected these toys were. Think of these toys as the "pixels" of future tangible interfaces. And think of the internet (today about 100m PC's, in a few years about 1b) as a foundation. On top of it, vast new "capillary" networks will sprout, connections for the real world.

This meeting is the first of a kind. Our purpose is to begin a systematic exploration of embedded systems. There is every reason to believe that in a decade or two, physical things (appliances, clothes, toys, factories, you name it) will form the dominant and natural interface to the digital world. Intelligence within, embedded or linked, will be the hallmark of these systems. This workshop is intended to begin strategic discussions to that end. It was triggered by the confluence of a number of events: percolation of recent research at the Media Lab; a push from the White House to spur the development of new and radically different capillary networks; impetus from DARPA to embed sensing and intelligence into our surroundings; numerous industrial currents (from Bluetooth to JINI and beyond); gregarious interest from USENIX; and more. We now need to develop the research agenda for what comes next.

This workshop was co-sponsored by USENIX and the MIT Media Lab and was organized in record time. Staff at USENIX and at MIT have been extraordinarily resilient. We thank them, and all of you, for your unrestrained interest.

Welcome to the beginning of a new era in systems architecture.

Sincerely,

Dan Geer, *VP and Senior Strategist, Certco, and Treasurer, USENIX Association*

Michael Hawley, *Assistant Professor, MIT Media Laboratory*

Co-Chairs, 1999 Embedded Systems Workshop

The Personal Node (PN)

Gregory G. Finn and Joe Touch
USC / Information Sciences Institute
{finn, touch}@isi.edu

Abstract

A Personal Node (PN) is a small, wallet-sized device that integrates people into the Internet. A PN incorporates wireless communication, limited user I/O, and local environmental telemetry to catalyze the coordination of other smart space (SS) and network devices for the user's benefit. By themselves SSs are not aware of the people in them and people are not be aware of what is in a SS. The PN allows the SS to interact continuously with a person, and a person to interact continuously with the space, mediating the interaction with the help of other devices throughout the system. A PN is an individual's networking focal point. As the user roams about, a PN persistently maintains user presence on the internetwork. This represents the final and missing link in SSs, bringing the user in as a system resource and participant.

1: Introduction

In the near future buildings, rooms and vehicles will evolve into *smart spaces* (SSs) containing varieties of wireless *smart devices*. As individuals roam they will swim in a virtual sea of such devices. Those spaces must become aware of the individuals in them and vice versa. A Personal Node (PN) is a small device that is continuously carried by an individual, **allowing the user to become a permanent part of the smart space**, and allowing the infrastructure ongoing access to the user for feedback and commands. The PN is the network corollary of the PC; it is a network node for the 'rest of us'.

A PN has promiscuous wireless interfaces to allow it to act as a networking catalyst. PNs also have telemetry sensors, such as location (GPS), temperature, and orientation. This allows a PN to be addressed by-interface, by-region, by-proximity or by-heading. Support for ad-hoc networks with multiple addressing modes benefits military, commercial and emergency services.

1.1: Need for a Sixth Sense

Wireless networks enable SSs and devices; key to the success of SS's will be their ease of use. People cannot directly interact with SSs. We are deaf, dumb and blind there, unaware of when we are in one or what's there. Conversely, SSs are unaware of our presence. This mutual unawareness must be overcome for SSs to be integrated into our daily lives.

In the distant future, SSs may have senses to be aware of and interact directly with us. It is currently easier for us to be aware of SSs and interact with them on their terms; to carry that sense with us. We need to acquire a sixth sense for us to continuously inhabit, to 'see and speak' in this wireless spectrum. One solution is the PN, to see and speak for us in the SS's wireless spectrum.

This document outlines our vision of the PN, and how it uniquely enables SS interaction via a continuous network presence for people. The rest of this document is organized as follows:

- Sec. 2: Defining a PN
- Sec. 3: Need for a PN
- Sec. 4: Research Issues
- Sec. 6: Related work
- Sec. 7: Implications
- Sec. 8: Summary

1.2: A vision of SSs

The following describes our vision of the future of SSs, and adding people to the continuous infrastructure. The vision is superficially similar to other SSs (e.g. Active Badges), but a PN is distinguished as the user's "eyes and ears" to the Internet (i.e. sixth sense). It is not a compute node, but rather the minimal I/O for a human to interact digitally.

> A user views a map in a monocular display. The monocular and its buttons provide primary I/O, while the PN provides location and orientation for the map to match the view.
>
> The user walks up to a truck, and the SS it signals a hand-off of the map view, to the heads-up display (HUD) or monitor at the seat the user enters, as he sits. The user drives off, and the map zooms out as the vehicle accelerates, to provide a more appropriate view.
>
> The user's rucksack, near his PN, is considered relevant by the truck's SS, and is checked for rations. A supply truck passes by, and the user's truck asks the supply truck for a refill of water, food, and batteries, which are dumped roadside with a smart, encrypted beacon. The HUD points to the refill pack as the user's truck approaches it, and the user's PN extends that function as a beacon-compass as

the user walks over to retrieve the sack. Many people in the same group have also recently retrieved rations, and that aggregate behavior schedules a shipment from the supply depot.

In this vision, the PN alerted the SS of the user's presence, and allowed another user (via the truck) to alert him an urgent query. This interaction also avoided external terminals, relying directly on the PN.

2: Defining a PN

A PN is a networking *vade mecum*[1] that is active whenever you are. It maintains your network presence, with enough I/O for bootstrapping other interactions.

2.1: Basic Assumptions

PNs require a number of conditions, most regard assumptions of Internet evolution, e.g.:

- Internet wireless services will be ubiquitous.
- Building nets will provide basic Mobile IP [18].
- Wireless nets provide different service levels.
- Smart-spaces will depend more on 'user state'.

2.2: Design Principle and Content

The main design principle of the PN is, **"have only those capabilities that cannot be moved elsewhere"**:

- User I/O to bootstrap user-network interaction.
- All the I/O that's particular to the user's locale.
- Support, i.e., wireless links, processing and memory for local operations and bootstrapping, a battery that recharges "when the user does".

A PN is a hand-held sized device, including:

- A variety of wireless interfaces:
  - Fast IR, for Mbps desktop roaming
  - Wireless LAN for 1 Mbps office roaming
  - Cellular for 100 Kbps MAN roaming
  - Satellite wireless for WAN roaming
- A limited amount of user I/O:
  - microphone, speaker
  - small LCD (PDA or smaller), buttons (3-4)
- As much telemetry I/O as possible:
  - orientation: GPS, accelerometers, compass
  - environment: temperature, humidity, light / IR, sound, camera, EMI, pH
  - personal biometry: pulse, respiration (via sound proc.), blood pressure, fingerprint

A PN is not a full computer. It does not include:

- File storage
- Traditional I/O (keyboard and display)

1. *vade mecum* (latin) - lit. "go with me"

A PN mediates for its user with SSs that it encounters, making these SSs aware of the user and the user's environment. Conversely, a PN becomes aware of what is in the SS environments. A PN also caches information of immediate or local relevance on behalf of the user.

To achieve these objectives, PNs must be carried by their users most of the time, like a pager or cellular phone. As a result, a PN must be small and lightweight; this precludes a standard keyboard or display. Unlike a wearable computer or laptop, a PN is not intended to replace a workstation [24]. We assume that a user's primary computing and storage is located elsewhere, and the PN acts as a bootstrap to catalyze their coordination.

Compared to a PDA, a PN is smaller, intended for simpler network-to-user interactions, but with more advanced sensors and network access. A typical PDA is the size of a deck of cards, and limits its I/O to a fairly large display, stroke-based text input, and audio output.

A PN contains I/O peripherals intrinsic to an individual that cannot effectively be located elsewhere. This includes user I/O, including microphone and speaker, control keys, and a limited display, eventually packaged the size of a large wristwatch (Figure 1). It includes sensors that monitor you and your environment, such as GPS, electronic compass, accelerometer, photometer, barometer and biometry. Unlike PDAs, PNs are IP-addressable, operate continuously and autonomously, and communicate with their wireless environment.

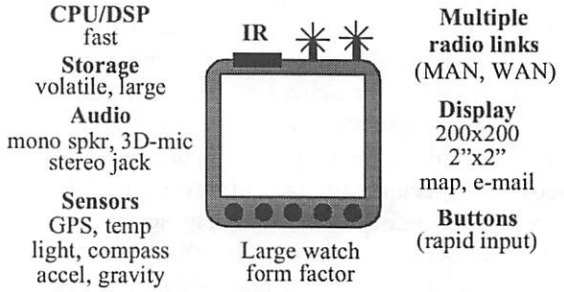

FIG. 1. PN characteristics

There are a variety of research issues in the PN. Primary is power conservation, which drives both hardware and protocols design. Continuous connectivity also challenges the traditional model of a node (host) [4].

2.3: How is a PN different?

The PN is related to the ParcTab, but differs as it:

- Supports continuous internetwork presence
- Is I/O rich: wireless, audio, and instruments

The PN extends SSs, enabling participants outside their conventional range. Continuous inclusion of the user as a node in the global infrastructure uniquely distinguishes the PN. These differences are summarized in Table 1.

property	PDA	PN
on time	mins/mo	100% on high-duty days/mo
usage	primary when disconnected	always secondary (catalyst)
num conns	1	2-3 (for handoff)
output	display, min. speaker	display, stereo spkr
input	keystrokes (primary) buttons, mic	mic (primary) buttons
other I/O	as peripherals	integrated GPS, temp, light accel...

Table 1: Comparing PDAs and PNs

3: Need for a PN

As noted, a PN integrates people with the network, enabling new uses for SSs. This section elaborates on the extension of SSs to enable new capabilities.

3.1: PN as a "Smart Spaces Device"

Since the early ARPANet, networking assumed two kinds of nodes - hosts and routers [4] [6]. This is modestly extended via Mobile IP and DHCP to support hosts that relocate periodically, such as laptops. IP telephony adds another node type, i.e. the IP telephone.

New devices extend this model further. The aggregation of host- and router-like devices is the basis of desktop area networks and network of workstations (DAN [1], Viewstation [10]). The current model of host as a single, terminal node is insufficient for these cases.

Even so, many emerging network devices remain expressible in this model. Wearable computers are modeled as laptops, since they not used continuously, and relocate only sporadically. They are alternate hosts, replacing a PC, but otherwise the same kind of node.

By analogy to telephones, the PN fills a gap in the node design space (Figure 2). Earlier attempts addressed this 'non-host, non-router' role [21]. In early telephony, limited resources drove party line use (e.g., mainframes with batch sharing). As telephones became less scarce, direct dial and home nodes resulted (PCs).

Users relied on telephones because they were pervasive in permanent places. This extended to mobile phones, initially by "move and reconfigure". In all these, messaging services were necessary, because the callee wasn't always available. Telephones were asynchronous, except during business hours.

Telephones evolved to true mobile phones and pagers. These are ubiquitous *vade mecums*, with a single, persistent user and number. Before, a phone number was a business or home. Now phone numbers are people. Hosts evolved from office to laptop addresses. The PN is the next step, allowing people to become nodes.

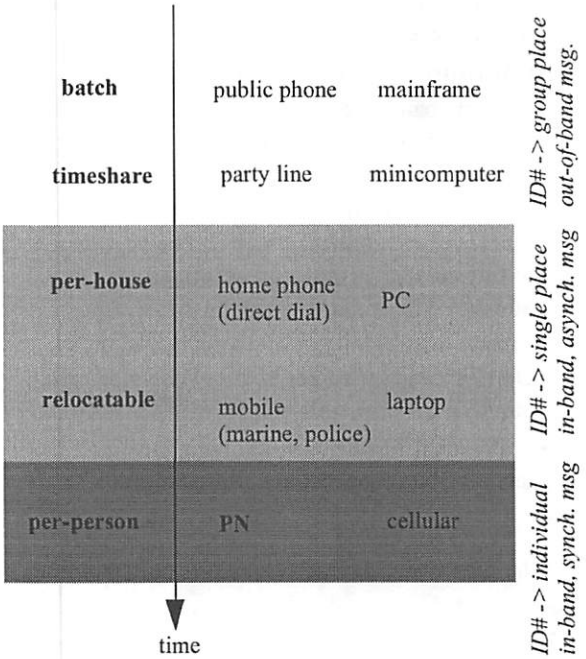

FIG. 2. Computer evolution emulates telephony

3.2: PN Application Domains

Behavior exhibits a phase change when accessibility is continuous, as seen in page or cell-phone use. The PN similarly is a catalyst enabling user interaction with SSs.

Presence Sensitive Applications

The field operative example shows an SS augmented by sensitivity to users. There are many other examples:

- Smart doors automatically unlock for users.
- A smart docent delivers user-specific tutorials based on the PN's data about the user's language and level of subject knowledge.
- Soldiers' PNs are asked for supply levels, which is aggregated for logistics support.
- A smart subway collects user destinations, to skip empty or unnecessary subway stops
- A PN is loaded with URLs on entering a smart business space. Current prices, private to the group, are provided to local PNs.
- Training scenarios gather the user status, to catalog participants and behaviors.

Smart Emergency Spaces and Services

A PN is a wireless point-of-contact for the user, with instrument and minimal processing assets. This combination enhances emergency services (ES):

- A PN can be loaded with site-specific ES information and software.
- A PN can authenticate ES messages.

- When local nets fail, a PN can use its wireless links for ad-hoc baseless networking.
- A PN can be addressed by location, finding a user or sending alarms to affected PNs

Autonomous Information Gathering

As a user roams, her PN will pass through SSs. Data can be cached by regional broadcast to passing PNs. Data can also be forwarded by the PN to the SS.

- PNs in vehicles or carried could indicate current level of supplies, which can be aggregated to logistics for disposition. Shortages can be addressed by redirection of the driver/user toward depots, while also debiting that depot.
- Delivery and repair services operate more efficiently. Resources are staged as needs move.
- A PN identifies its user, allowing businesses to use profiling to direct the user to specials.

4: Research Issues

PNs raise a number of research issues that concern protocols, naming and addressing, coordination and configuration, privacy, scale and user interface. Their existence enables new application areas that involve roaming, information capture and ad-hoc networking.

4.1: Protocol integration

The PN requires continuous communication as it spans different link technologies, and may require periodic hibernation to conserve power. State-oriented network protocols, notably TCP, do not react well to such idle periods and are not intended to support endpoint renaming. TCP is inefficient for simple request/response protocols. T/TCP is better, but requires further modification to support seamless transitioning [5].

The PN may also need to support proxy operation of other protocols. These might include delegated request/response, switchboarding, or remote control of a delegation point that coordinates or redirects multimedia streams among other SS devices.

Device-Control Protocols

Smart devices and PNs are network attached peripherals (NAPs). NAPs are uncommon today and most utilize media-specific transport protocols. Making smart devices themselves internet-accessible raises two issues:

- Standardization of device network APIs [9]
- Preservation of privacy and integrity [22]

Link agility

To maintain network presence the PN must monitor the 'liveness' of its links and reassociate with new ones as needed. Liveness monitoring and link association algorithms need to be developed. Low-power consumption is a requirement that must guide this work. There are a number of triggering events to consider:

- Failure of 802.11 low-power beaconing
- Hearing mobility agent from another subnet
- Distance from wireless hub
- Rate of damaged messages
- Heading and speed

Proxy protocols

The PN is not necessarily the best device to run request/response or stream protocols. Its limited bandwidth and power hinder it from first-class participation as a router in the SS. Instead, it may be appropriate to off-load some protocols to proxies at other SS devices.

Protocol trade-offs

There are a number of bandwidth vs. latency vs. power trade-offs in a PN that need to be considered:

- Asymmetric protocols to reduce transmission
- Periodic retransmission and server anticipation
- Split interfaces, i.e. IR out, 802.11 in

4.2: Naming and addressing

The PN challenges the traditional network notions of naming and addressing. The name of a PN should be intrinsically linked to its owner, not the device itself. GSM cell phones have this property; the encoding card identifies the telephone number, independent of phone. Other challenges are only beginning to be researched.

Geographic Addressing and Broadcasting

Providing each PN with knowledge of its location allows it to be addressed both by-interface and by-location. Various approaches to realizing geographic routing and broadcasting need to be examined.

Geographic routing and addressing in the Internet has been approached by creating a virtual network from geographically aware routers located within the Internet [15]. Earlier work pointed out that geographic addresses could be used directly by routers [8].

Geographic addressing could be grafted into IP as an option. However, IPv6 reserves 1/8th of its 128-bit address space for geographic addressing [19]. The possibility of providing hosts both interface-oriented and location-oriented addresses needs to be investigated.

SS Discovery

As a PN enters a SS it should become aware of that SS and vice versa. Mechanisms are needed for a PN to discover available smart devices in a SS, which ones it can access, and the application interfaces they support.

Generalizing Multicasting

Multicast groups are currently created and associated with a multicast address. A host explicitly joins a group to become its member and routers alter their routing tables to service group members. Once joined, a host remains a member until it explicitly leaves the group or

the group is destroyed. Host mobility has no effect on membership in this type of group.

A multicast group could also be defined geographically, e.g., hosts within a geographic region, a room, a building. To within location precision, a host is inside or outside a group's region. Membership in a multicast group is then implicit, regardless of host mobility.

There are significant differences in how these two classes of multicast group are defined and in how membership is defined. Mobility makes ephemeral the membership criteria in geographically defined multicast groups. The set of such multicast groups that a mobile host is potentially a part of could change frequently.

4.3: Coordination and configuration

The determination of resources in a SS that a PN has entered and the run-time matching of those to application requirements is a producer/consumer problem that requires solution.

In conventional system architectures, both device controllers and devices are resident in the chassis. However, in a smart-space environment the set of devices that a PN may come into contact with as the user roams will be large and dynamic. It will be unrealistic to expect a PN to be preconfigured with all needed drivers. This immediately imposes requirements upon PNs:

- Need for Dynamic Reconfigurability
- Interface Matching
- Push/Pull Configuration Software

4.4: Scale

Both PNs and SSs increase the scale of networks in a variety of ways. The dynamic range of bandwidth increases, mostly due to a lower bandwidth required for WAN signals to the PN. The latency range increases, also because WAN signals are liable to use satellite paths. The number of devices in the network vastly increases because addresses are required both for SS components for PNs. While SSs might support address aggregation, the global roaming capability of PNs may inhibit such a simplification.

4.5: Security

The need for security is especially important for PNs because they are directly associated with individuals, and because they contain so much local state (GPS, microphone, etc.). Two levels of security are required for PNs. The data content itself must be secure (authenticated or encrypted), and the event of communication may also require privacy (source confidentiality) to prevent tracking or behavior monitoring.

There is an external aspect of security, that of device theft, which also requires consideration. The small size of a PN encourages its theft. A number of approaches deserve consideration:

- PIN at power-up and periodically
- Secure card for sensitive state like PCS phones
- Biometrics: voice, fingerprint

4.6: User interface

Size limitations prevent a PN from having a traditional keyboard or display. It is also unrealistic to expect individuals to use a conventional display and keyboard while roaming. It is our contention that when users are not roaming they will likely be able to use nearby conventional display/keyboards. Consequently, the user-interface for a PN must be nearly hands-free, depending primarily upon voice control with limited use of buttons.

Developing an effective nearly hands-free user interface for roaming will be a major research area. The need for speech recognition does imply that a PN needs access to significant computing resources.

5: Feasibility

The PN is currently feasible, even given our demanding combination of capability, portability, and continuous operation. It is possible to bootstrap its development with an off-the-shelf, rapid prototype, in parallel with the coordinated application of well-known low-power, integrated packaging design.

5.1: Rapid prototyping

The principal components needed to prototype a PN are readily available. Setting aside the size and packaging issues, much of the needed research and development could be done using a prototype built from off-the-shelf components. Existing PDAs and handheld PCs, together with wireless PCMCIA network interfaces can be used to emulate a PN.

Once that prototyping effort is finished, packaging, size and power consumption issues could be addressed separately. The industry has already developed suitable low-power processors, memories and interface circuits.

5.2: Power conservation

PDAs achieve their month-long recharge intervals by remaining normally off. They await an explicit activation by the user, perform a small amount of resulting computation, display the result and then turn off again.

On the other hand, PNs must maintain persistent network presence and so cannot be turned off. PNs must achieve a minimum recharge interval of 24 hours.

Variations on paging and polling techniques let a PN approach the normally-off power consumption characteristic of a PDA without sacrificing its network presence at the cost of increased latency. Each technique requires the cooperation of wireless hubs.

- Hubs page a low-power pager in the PN.
- PNs poll their hub (simpler, scales worse).
- Hubs poll the PNs (802.11 low-power mode).

Allowing a PN to remain mostly inactive will extend battery lifetime to a reasonable recharge interval.

5.3: Communication

The PN requires integration of multiple link technologies, from high-speed desktop to low-speed wide-area. Current variants of these include FIR (fast InfraRed) for the desktop, 802.11 wireless ethernet for the office, Metricom for the city, and text paging for the wide-area.

These different technologies have widely varying bandwidth, latency, and reliability. The application protocols need to adjust to available link capability, operating in loosely- or tightly-coupled modes as needed.

5.4: Packaging

Packaging issues include power conservation, thermal diffusion, ruggedness, and integration for miniaturization. Current sensor technology already supports component-level and chip-level versions of many of the devices proposed for the PN. Power consumption and heat buildup can be reduced by conventional techniques (power devices only when in use, or only periodically).

There appears to be a common *vade mecum* size, that of a cell phone or PDA, that is acceptable. A PN therefore needs to be at most wallet-sized and 1 lb.

6: Related work

The PN is a variant of PDA technology and wireless 'presence' devices. It also extends the networking efforts of recent wireless and mobile protocols.

6.1: Handheld PDAs

Recent PDAs and handheld PCs have integrated small displays, touchscreens, and sometimes keyboards to provide access to their limited local resources. Examples include the 3Com Pilot and a variety of WindowsCE palmtop PCs. The Philips Nino incorporates a microphone to support speech-based commands [17]. In addition handwriting or script recognition is provided. Compared to the PN, these lack environment sensors, have only limited wireless capability (usually only IR) and are used intermittently.

6.2: Wireless 'presence' devices

The ParcTab [23] was an early example of a wireless personal node. It had a single infra-red interface, provided persistent presence for in-building roaming and provided PDA-like services directly. In contrast, the PN extends wireless access beyond the building confines and focuses on catalyzing of other devices on its behalf.

The Lovegety is a simple wireless personal node that demonstrates mobile information capture [12]. It uses peer-to-peer beaconing, is small enough to fit on a keychain and runs for days to weeks on battery power. It can be considered a low-bandwidth variant of a PN, enabling singles to detect each other within a SS created by a multi-party ad-hoc baseless network.

6.3: Wireless and mobile protocols

Providing persistent internetwork presence while roaming is the subject of Mobile IP [18][13]. A mobile computing environment that uses multiple types of wireless networks is called a *wireless overlay*. Maintaining connectivity while running applications in this environment is extremely challenging [14][20] and is the subject of a number of research efforts, including BARWAN [2] and Odyssey [16]. Operating systems originally designed for workstations require extensive changes in a nomadic environment [3].

Imielisnski and Navas proposed embedding of geographic routing and addressing in the Internet by creating a virtual network from geographically aware routers [15]. Geographic addressing can also be directly used to route packets, support host mobility and provide regional broadcast [8].

7: Implications

The PN extends and challenges SSs and general network research. By including users as nodes, it extends the scope of the network, and the capability of applications. It also uses technology being developed for low-power, integrated sensors in a unique way to provide ad-hoc mobile smart sensor nets.

7.1: Smart spaces

The PN avoids the distinction between a user's on-line and off-line presence. The user is always on-line, accessible to signal for feedback, supporting immediate urgent-mode interaction. Because of its integrated, multi-level links it helps catalyze the aggregation of other network resources for the user's benefit.

7.2: Network research

Traditional networking considers users as temporary presences at permanent end-points known as hosts. The PN extends this notion, where people themselves become end-points on the network. People are more mobile, even than laptops, and so require hand-off without dead time, and a truly persistent identifier. Location of a user is a key network resource discovery issue.

The PN provides an opportunity to review more conventional host and gateway requirements, using a model that challenges their assumptions. Overall network architecture, naming, addressing, and resource discovery all may require re-examination.

In addition, transport protocols may require additional support for continuous relabeling of the end-points, as users move between SSs. The PN itself may provide bridging capabilities between adjacent PNs when necessary. Finally, the traditional request/response

protocols may require redesign, to support a proxy-mode operation, to off-load capabilities to SS resources and conserve local power.

7.3: Application of related technology

There are a number of related technologies that are required for a PN to be developed. Small, low-power sensors already exist, but need to be integrated with a small amount of processing and storage into a handheld device. The PN focuses on placing as much I/O technology where the user is as possible, so there is virtually no limit to the challenge to integration technology here.

By placing the sensors where the user is, the PN provides a unique opportunity for ad-hoc deployment of sensor networks, in effect a mobile SS centered, and concentrated exactly where the users are. Deployment at the correct place is de-facto achieved.

There is also a challenge to integrate a number of wireless communication technologies into a single, low-power, configurable device. This includes bandwidths from 1-1Mbps, latencies from ms to 100's of ms, and ranges from feet to tens of miles, using IR, CDMA, GSM, and even simple analog paging technologies.

8: Summary

Wireless technology will soon be used to create and leverage SSs comprised of peripheral devices and sensors that communicate with one another and the network. As humans we can't directly perceive the wireless spectrum and so we aren't aware when we are in a SS and we won't know what's in it. Conversely, SSs won't be aware of our presence. As long as people do not have the capability of directly interacting with SSs as they roam, SSs will remain restricted in their scope of application and ease of use.

These issues are addressed by creating small personal wireless nodes (PNs) that are carried with individuals as they roam. The PN's goal is to integrate the human being into the Internet. The PN allows the user to become a persistent part of the network, by providing:

- - continuous communication with the user
- - user-centric telemetry and biometry sensors

By providing a minimal initial access, these capabilities can be used to bootstrap the user's access to more advanced services, and to support ad-hoc base-less networking when disconnected from the rest of the net.

The authors would like to thank Bill Manning, Gene Tsudik and Jon Postel for their helpful suggestions. An earlier, extended version of this paper appeared as an USC/ ISI Research Report, ISI-RR-98-461, July 28,1998.

[1] Barham, P., Hayter, M., McAuley, D., Pratt, I., "Devices on the Desk Area Network," IEEE JSAC, May 1995.

[2] BARWAN Project, http://www.cs.Berkeley.edu/~randy/Daedalus/BARWAN/BARWAN_over.html

[3] Bender, M., et al., "Unix for Nomads: Making Unix Support Mobile Computing," First USENIX Symposium on Location Dependent Computing, 1993.

[4] Braden, R (ed.), "Requirements for Internet Hosts - Communication Layers," RFC1122, Oct. 1989.

[5] Braden, R., "T/TCP -- TCP Extensions for Transactions, Functional Specification," RFC1644, Jul. 1994.

[6] Braden, R., Postel, J., "Requirements for Internet Gateways," RFC1009, Jun. 1987.

[7] DIGITAL Semiconductor SA-1100 Data Sheet, EC-R8XUA-TE, Digital Equipment Corp., Jan. 1998.

[8] Finn, G. G., "Routing and Addressing Problems in Large Metropolitan-scale Internetworks," Research Report ISI/RR-87-180, USC/ISI, Mar. 1987.

[9] Hotz, S., Van Meter, R., Finn, G. G., "Internet Protocols for Network-Attached Peripherals," Sixth NASA Goddard Conf. on Mass Storage Sys. and Tech., Mar.1998.

[10] Houh, H., et al., "The VuNet Desk Area Network...," IEEE JSAC, May 1995.

[11] Imielinski, T., Navas, J. C., "GPS-Based Addressing and Routing," Tech. Report (LCSR-TR-262), Rutgers Univ. Computer Science, Mar. 1996 (updated Aug. 9, 1996).

[12] Lovegety, http://www.freeyellow.com/members2/onmarketing

[13] Johnson, D. B., "Mobility Support in IPv6," (work in progress).

[14] Katz, R., Brewer, E., "The Case for Wireless Overlay Networks," Ch. 23 in *Mobile Computing*, Kluwer Academic Publishers, 1996.

[15] Navas, J. C., Imielinski, T., "GeoCast: Geographic Addressing and Routing," Third ACM/IEEE Int'l. Conf. on Mobile Computing and Networking (MobiCom'97), Budapest, Sep. 26-30 1997.

[16] Odyssey Project, http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-ody.html

[17] Nino PDA, http://www.nino.philips.com

[18] Perkins, C., Editor, "IP Mobility Support," RFC2002, Oct. 1996.

[19] Rekhter, Y., Li, T., Eds, "An Architecture for IPv6 Unicast Address Allocation," RFC1887, Dec. 1995.

[20] Satyanarayanan, M., "Fundamental Challenges in Mobile Computing," Fifteenth ACM Symposium on Principles of Distributed Computing, May 1996, Philadelphia, PA.

[21] Stewart, B., (chair), "Charter of the Special Host Requirements (shr) working group", Proc. 21st IETF, Jul. 1991.

[22] Van Meter, R., Hotz, S. Finn, G. G., "Derived Virtual Devices: A Secure Distributed File System Mechanism," Fifth NASA Goddard Conf. on Mass Storage Systems and Technologies, Sep. 1996.

[23] Want, R., Schilit, B.N., et al., "The ParcTab Ubiquitous Computing Experiment," Ch. 2 in *Mobile Computing*, Kluwer Academic Publishers, 1996.

[24] Wearable Computing FAQ, http://lcs.www.media.mit.edu/projects/wearables/FAQ/FAQ.txt

Discourse with Disposable Computers: How and why you will talk to your tomatoes

David Arnold, Bill Segall, Julian Boot, Andy Bond, Melfyn Lloyd, Simon Kaplan

CRC for Distributed Systems Technology (DSTC)
The University of Queensland, St Lucia, 4072, Australia
Phone: +61 7 3365 4310 Fax: +61 7 3365 4311

{arnold,bill,julian,bond,melfyn,simon}@dstc.edu.au

Abstract

Beyond ubiquitous computing, is the advent of disposable computing, occurring when the price of an embedded computer becomes insignificant compared to the cost of goods. Current software and network architectures and their associated programming paradigms will not scale to this new world. The necessity of catering for the constant change in number and type of devices of interest to a user, as well as their sheer quantity, dictates new approaches to construction of software systems based on more flexible models.

We propose that distributed event notification forms a fundamental requirement for systems of this scale, and discuss the advantages of undirected communication over current interaction models. Our experience with Elvin, a prototype notification system motivates the discussion and serves as illustration of its possibilities.

1. Introduction

We are rapidly approaching an era where most consumer products contain an embedded computer and network interface. While the availability of ubiquitously "wired" goods is currently a novelty, it will soon be not only commonplace, but all pervasive.

However, we contend that most predictions of ubiquitous computing drastically understate the number of networked devices. While it is easy to imagine networked toasters, fridges, televisions, and indeed, any already electronic device, following these will be the second wave of wired devices; the era of *disposable computing*, when the price of embedding a computer becomes insignificant compared to the cost of manufacture. Far more than ubiquitous computing, disposable computing will wreak fundamental changes in the nature of computing, allowing almost every object encountered in daily life to be "aware", to interact, and to exist in both the physical and virtual worlds.

In particular, disposable computing dictates a new approach to interaction amongst software components, and between software and human users. The issue facing software architects is how do we effectively use networked food, clothing, paper, books, people, doors, cars and roads? What communication strategies are needed? How do we manage quadrillions of devices? And how do they interact with us?

We begin by describing some properties of future networks, and a scenario that drives an analysis of requirements for computer-enabled interaction on a vast scale. A prototype system for pervasive, contingent component interaction is introduced, and discussed in light of the scenario. Some of our current endeavors indicate useful properties, and we discuss some of the many research challenges that remain the subject of future work.

2. The Wired World

For over a decade, computer scientists have predicted the integration of computers and networks with the affordances of our daily life [Wei91]. The development of hardware has today reached a point where this is technically viable, and it will shortly become financially accessible to average consumers.

The software challenges offered by the previous generation of hardware are being answered by technologies like Plug and Play and Jini, but the grand challenges of ubiquitous computing remain unanswered. As we examine interaction models for software, we consider four particular problems and their impact.

The Quadrillion Node Net

The first wave of consumer electronic devices with a network interface will extend the current global network to trillions of devices. But it is the second wave, the instrumentation of non-electronic devices, which ushers

in the Quadrillion Node Net. When every book, packet, street sign, soda can and pen is active and networked, the number and diversity of devices challenge out ability to control and manage them.

Disposable Computing and Device Churn

How often do you buy a new computer? And when you do, how long does it take to get it set up the way you need it? When every manufactured product you see larger than a paper clip is a computer, how do you configure them? Rather than acquire a new computer every year, you will acquire them every minute, sometimes by the 1000. And you will throw or give away computers at the same rate (or your partner will finally leave you!). Objects with embedded computers will appear and disappear from the containing network at a frantic rate.

Security and Charging

When you throw you lunch wrapper in the trash, its computer negotiates with the trashcan to be recycled or shredded or composted. But your lunch wrapper was bought using your debit account, and the trash can wants to charge you for burdening it with non-recyclable plastic...

The possibility for eavesdropping and losing sensitive information becomes overwhelming once computers are disposable. The volume of data available about you and your life becomes absolutely staggering. How do we secure your information environment whilst retaining the availability and mobility of your data? How do we balance the benefits of availability whilst protecting against intrusion.

Context Management

Software components are remarkably good at ignoring unwanted stimulus, but people become quickly irritated by untimely information. The benefits of having the universe at your fingertips are quickly overlooked if the universe is always in your face. When you are responsible for a million interaction-rich computers, these interactions are going to need to be coordinated, filtered, and exchanged, but above all mediated automatically.

Users must be able to set policy for their interactions with the environment that includes the context, not only of themselves, but their interactions with other objects at any given time. Context management encompasses the mechanisms used to specify what is appropriate user interaction, and to automatically determine when and how it is appropriate.

A common, vital element in the solutions to these problems is the nature of communication between software components. Distributed systems currently use a variety of protocols, with a growing general reliance on an RPC-style model. However, RPC and remote method invocation are constrained to a request/reply interaction, using known interfaces types at a specific, possibly indirectly resolved, address.

But the universe of disposable computing is populated with devices whose type and identity are completely unknown to the other devices they will have to interact with. The continual churn of artifacts relevant to a task will completely overwhelm our current solutions of name servers and well-known addresses within the homogeneous IP network.

The next section introduces the scenario that the rest of the paper uses as the basis for analysis of these issues, and presentation of a possible solution.

3. Pasta, circa 2005[1]

Somewhere in Germany there is a factory that produces the little cans that canned food goes into. This factory makes cans that appear perfectly normal it's just that each can contains a tiny computer, a small amount of memory, and a short-range radio transceiver. It's a smart can and the factory that makes them charges eight pfennigs more for each one. As part of their production, the cans get embedded with a small amount of data such as the date of manufacture, the batch and can number, the alloy details etc.

Once produced these cans travel all over Europe. One batch of these cans is sent to Italy where they go to a tomato-canning factory and are filled with tomatoes. At this factory, as part of the canning process, the can gathers a little more data: it is full of diced Roma tomatoes, it was filled on a certain date as part of a particular batch, and it has a particular use-by date.

One of these cans of tomatoes gets exported to the USA. As it moves off the wharf it is processed and its data content is translated from Italian to English. After a brief stint in a warehouse it ends up on a supermarket shelf. At the supermarket it inherits a little more information such as the retail price and date of being placed on the shelf. At some point a customer's pantry knows to order the can and one is sent to your house in the next delivery. Before the can leaves the store, the supermarket extracts the information it needs for stocktaking.

Some weeks later you're at your desk at work thinking about dinner, and decide that tonight you're going to cook a romantic meal for two. You look up your recipes, select one, and check your pantry for the necessary ingredients. Your tomatoes have cheerfully registered themselves to the pantry upon arrival, so it is able to

report that all you need is some fresh basil that you can pick up on the way home.

At the supermarket, you find the basil and drop it into the trolley, which updates the cumulative price of your selections. Noticing the screen's flicker, you glance down and see an advertisement for a special on oregano. You cancel it and disable further advertising.

Finally done, you push the trolley through the checkout, where your account is debited for the total, and your home address attached to your items. You push the trolley onto the track for delivery before heading to the cafe for a coffee on the way home as the store delivers the shopping for you.

At home you begin to cook, placing the opened can of tomatoes from the pantry onto the table. The can reports that it has been opened (after detecting the pressure differential).

You've been meaning to get the auto-light on your gas stove fixed for weeks now and seemingly every time you want to light it you can't find the matches. You ask the kitchen to locate the nearest box for you: there's one in the cutlery drawer. You've had enough though, so you direct the kitchen to factor the stove repair into your budget. Your stove knows not to hassle you again.

Having enjoyed your meal, you turn on the television but during the first ad break a scrolling message from the kitchen appears at the bottom the screen telling you that there's an open can of tomatoes that's been getting warm for over two hours. You swear briefly, but are at least glad the house didn't interrupt while you were busy. It knows you're not watching an important show and it did have the decency to wait for an ad break. You go to the kitchen and put the can into the fridge, pausing briefly to put the matches back on the fridge where you expect them.

Three days later you wake up and struggle to the kitchen for a cup of coffee. As you grab the milk, you see the fridge's display panel has a number of messages for you. You'll deal with the emails later but notice that the fridge is complaining that there is a can of tomatoes that is getting beyond its prime. At first you can't find them, but the fridge locates them behind the last of the beer, and you grab the can and blend them. Enjoying your tomato juice with your coffee, you begin a casual cleanup and throw the empty can into the recycling unit.

The recycling unit strips any personal information from the can, and noticing the alloy content ensures it gets picked up for recycling. Some time later the can is shipped to Germany for recycling.

1. Inspired by Hiro's pizza box in Neal Stephenson's *Snow Crash* [Ste92].

4. Disposable Interaction

Examining this scenario, and the state of hardware technology today, it seems that the production of such processors and network interfaces is practical, if not yet commercially viable. The wide range of devices involved, from the smart can to the local supermarket's CPU cluster, might require a heterogeneous network, with the peripheral processors using different protocols (and physical media) to the Internet backbone. We assume that arbitrary connectivity is feasible, with the possible use of proxies or gateways as required.

Given that this is the case, our current interaction paradigms could, by simple extension, support the proposed scenario. Or could they?

Messaging, RPC and multicast can all be termed *directed* communication models: the destination of the message is specified at the time it is sent (in the case of multicast, this specification is not a single address, but a group or channel upon which the senders and receivers have previously agreed). The problem with requiring knowledge of the destination is that sometimes you don't have it, and this has led to the development of numerous methods of obtaining addresses

- use standardized names, a name server, and a reserved address for local name servers, ie. [GA090], or
- use LAN segment broadcast or a reserved multicast address to find named objects, ie. [CG85], or
- use a yellow pages service at a reserved address, and select one of the available services in the required class by its advertised properties [OMG97], or
- perform a multicast request to a reserved group, and have all services listen to that group and respond if they can provide the requested function [VGPK97], and work in progress on [GPVD99]

This list is only superficially representative; resolving addresses for directed communication has absorbed a great deal of distributed systems research over the past decade. And yet none of these approaches really solve the problem. Each of them merely shifts the required knowledge to a level of indirection, without addressing the basic issue: that the originator of the message must know where it is to be sent.

In a system where we seriously expect quadrillions of computers, and several orders of magnitude more active

endpoints (or objects), and where the set of these relevant to an individual is in constant flux at rates of up to hundreds per second, requiring that the sender of a message always specify its destination does not appear feasible.

We propose an alternative that will exist alongside directed communication to ameliorate this problem: *undirected communication* is that where the sender of the messages does not specify their destination.

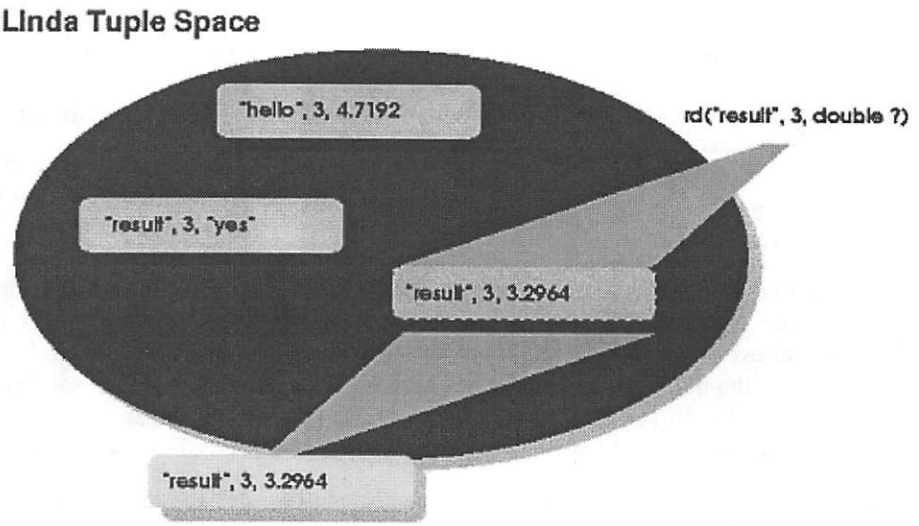

Figure 1: Linda's *rd()* copies a tuple matching the supplied template.

How can this work? By using a "pull" style, content-based selection of messages. Content-based addressing is not new. It has been widely used in specific applications, and was first popularized (to our knowledge) as a general communication mechanism by Gelernter's Linda [GB82]. It can easily, if inefficiently, emulate directed communication, leading some to propose it as a universal communication model. We prefer to use it in conjunction with directed forms of communication, selecting the model most appropriate for the task at hand.

For content-based addressing to work, message *consumers* (destinations) must have a way to specify that they want to receive a certain class of messages. This information is then used by the infrastructure to route the appropriate messages to the consumer. For the consumer to select a message from a producer (or source), it must somehow describe the message it is to receive. If this description is reduced to its simplest form, it effectively becomes a multicast address: a single, unique attribute used to identify a class of messages.

But using a single, unique attribute to identify messages offers no advantage over directed communication. While ultimately the consumer must share some knowledge with the producer(s), this knowledge can be structured to provide a flexible means of identifying pertinent messages by specifying selection criteria expressed in terms of the message's contents.

In Linda, these specifications are called *templates* and they describe the number, type and order of the message's attributes. The value of a particular attribute can be fixed by providing a value, or is otherwise constrained only to the required data type.

Notification services also provide a degree of undirected communication. Unlike Linda, notifications are transient, and without Linda's requirements for persistence, notification services scale to support a much greater overall bandwidth. MIT Athena's Zephyr[DEFJKS88] was followed by PEN[DB92], Rendezvous[OPSS93, TSS95], Keryx[Low97], Elvin[SA97] and others in this general domain.

In the terminology of Rosenblum and Wolf[RW97], the directed-ness of notification forms the *naming model*, where classes of events are named using either a structured name, or a property-based name. The degree of direction extends from a multicast address (very directed), through a filter-able structured name, to a property-based query (least directed).

Channel-based services use structured naming. While requiring producers to nominate a specific channel (often a hierarchical name of the form *foo.sub-foo.sub-sub-foo*), they typically allow wildcard filtering of channel names, and often some local secondary filtering of other distinguished attributes.

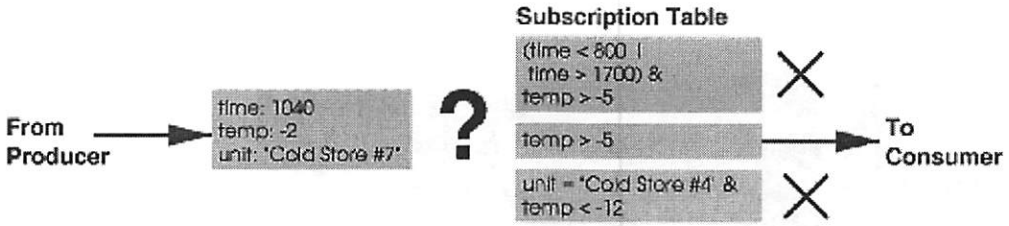

Figure 2: Evaluation of subscription expressions.

Keryx and Elvin (described more fully in the following section) use a boolean constraint language to select messages by their content. The messages are *self-describing*, with unordered attributes identified by name, and having a strongly typed values. They allow, for example, selection using numeric ranges and regular expressions on string values. While this mechanism still requires that the message producer and consumer are coupled by the definition of the attribute names, it is significantly more flexible than the other schemes. This has a number of practical benefits for distributed systems.

The deployment of distributed systems is hampered by the close coupling of components through rigid interfaces. Direct, point-to-point binding of components inhibits runtime substitution, removal or addition of components. Using undirected communications, components can be introduced or replaced without affecting any others.

In addition to limiting the interaction architecture of distributed systems to a client-server paradigm, the static definition of component interfaces using an IDL (ONC[Sun88, MS91], DCE[SHMO94], CORBA [OMG91], DCOM[Tha99]) severely restricts the ability of applications to adapt to changes in their environment. An endpoint is bound directly to a component, and cannot be implemented by a group of cooperating objects nor can components simply extend their functionality to include new behavior. Their API effectively dictates the structure of applications.

In a world of disposable computing, where the applications architecture must adapt to the constantly changing environment, interfaces must be able to split and merge, run on a single machine or be spread across the world. Running applications must be able to constantly and seamlessly *adapt* to their current context. And the use of directed communications makes this all but impossible.

The next sections discuss the Elvin architecture and implementation in detail, describing both its current form and the work currently under way to extend it to provide a ubiquitous content-based routing infrastructure for disposable computing.

5. Elvin Architecture

Elvin is a content-based message routing system under development at DSTC. It provides undirected communication, using content-based subscriptions to route self-describing messages.

5.1. Overview

In essence, Elvin routes undirected, dynamically typed messages between producers and consumers. Messages consist of a set of named attributes of simple data types. Consumers subscribe to a class of events using a boolean subscription expression.

Elvin can be described as a pure notification service [RDR98]. Producers push messages to the service, which in turn delivers them asynchronously to consumers. When a message is received at the service from a producer, it is compared to the registered subscription expressions for all consumers and forwarded to those whose expressions it satisfies (see figure 2). Elvin is a dynamic system: messages can be sent without pre-registration of message types and subscriptions can be added, modified, or deleted at whim.

The system is implemented as a server daemon that provides the subscription registry and evaluation engine. Client libraries map the wire protocol to programming languages. As well as workstations and personal computers, we are starting to experiment with devices like Palm Pilots and PIC/AVR-class embedded micro-controllers, using radio, IR and wired serial communications to the server.

The flexibility of distributing events based on content is often sacrificed by notification services due to a perceived lack of efficiency [WWWK95]. Common alternatives are to use named channels [DEFJKS88, RBM96, OPSS93, TSS95] or event types [OMG98, Sun99] that must be specified by both producer and consumer. A key benefit of content-based addressing is the reduction of this coupling between producers and consumers. A producer in a channel-based system must

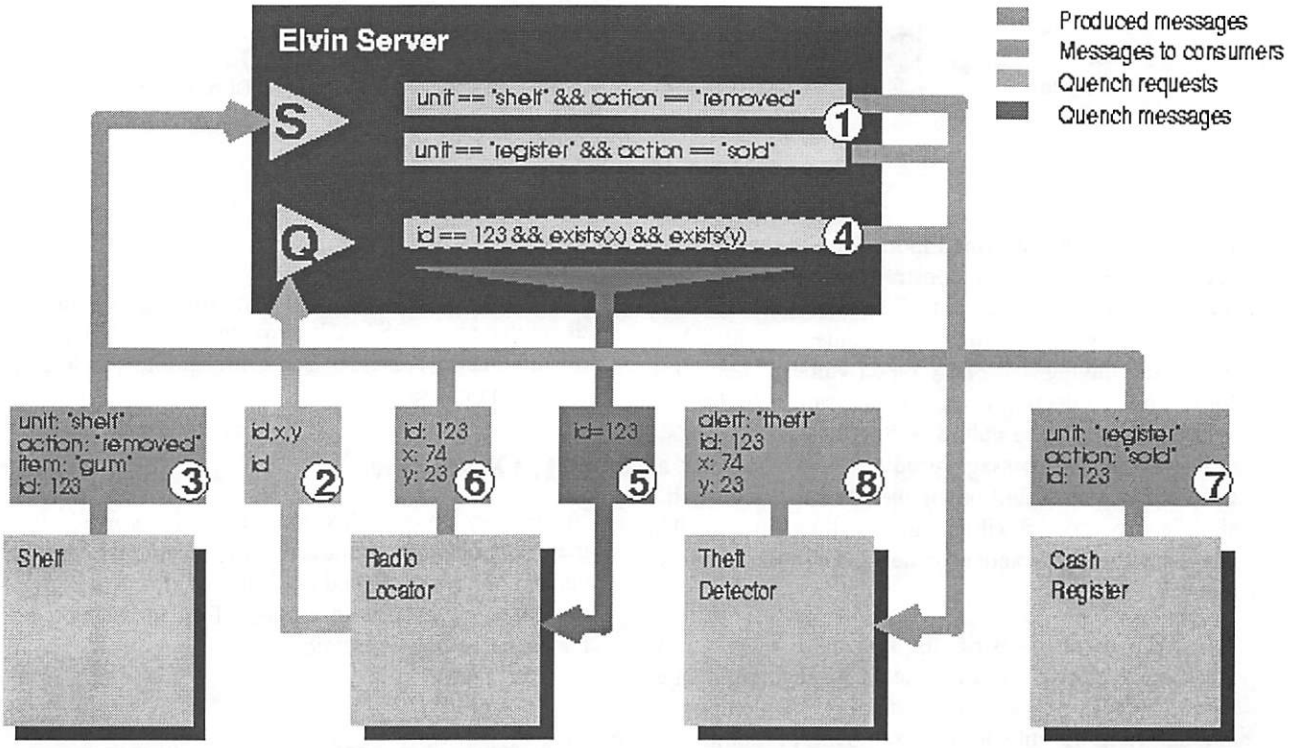

Figure 3: Using *Quench* to control message generation.

be made to send to multiple channels if more than one class of consumer requires the event. Content-based addressing allows any number of different consumers, including those previously unknown, to receive information based on what they need, rather than where the information was directed.

Once producers are freed of the responsibility to direct communications, the determination of the significance of message becomes less important: they can promiscuously send any potentially interesting information, and rely on the system to discard messages of no (current) interest to consumers.

5.2. Quenching

While decoupled message production and consumption is useful, situations where the cost of message generation is significant or the volume of traffic very large, require a "back channel" from the consumers that can be used by producers to determine interest in classes of messages.

The Elvin *quench* facility (named for its ability to reduce message traffic), enables producers to be told when a consumer (or consumers) has subscribed to messages with particular attributes, and optionally obtain the range of values requested. The producer specifies the attribute names that must be present in the subscription expression and the names of attributes for which they want to know the set of requested values. This information is forwarded to the producer whenever changes to the server's subscription base alter the specified values. The quench facility is thus effectively a subscription to messages describing changes to (or initial state of) an Elvin server's subscriptions.

Consider a producer that emits a large number of messages that at any given time might not be of interest to a consumer. By examining the registered subscriptions, it can determine when its information is of interest to a subscriber (or many subscribers) and control its emission.

Alternatively, if it is too expensive to generate unwanted messages, the quench facility can control generation. In the scenario from section 3, consider the supermarket and some packets of chewing gum: the gum is very cheap, so cheap that the manufacturer can only afford to put passive location tracking in the packaging. However, chewing gum is a prime target for

shoplifting, so the store wants to track the packets to enable them to detect attempts at theft.

Of course, there are thousands of similar packets in the store, and tracking each of them is well beyond the capacity of their radio location system. Fortunately, only a relatively small number of those packets are removed from the shelves at any one time. What is required is a mechanism enabling the location tracker to determine which packets are of interest.

In figure 3, the theft detector has registered two subscriptions: one for removal of items from the shelves, and another for the sale of items from the cash register (step 1). The radio locator requests quench information for subscriptions to location events (2). After being notified by the shelf that a packet of gum has been removed (3), the theft detector subscribes to notifications of its location including the unique identifier for the packet (4).

The radio locator needs to know what items to track, without directly coupling it to the theft detector (or any other system requiring location information). It needs to examine the active subscriptions to determine for which items location events are of interest. The theft detector's subscription (4) matches the quench request from the radio locator (2), and the `id` attribute value is forwarded (5). The radio locator begins tracking the gum, and emitting location messages (6).

Finally, either the gum is sold, and the cash register's sale message (7) informs the theft detector that it need no longer monitor the item, or, if the location coordinates move outside an approved range, the theft detector can emit an alarm (8).

Using the quench facility in this way, producers are able to determine consumers' requirements without losing the flexibility that the decoupling of message production from consumption gives.

6. Elvin 3

Elvin3 is a publicly available implementation of the Elvin architecture, and has been in use for nearly two years. It uses a single TCP/IP-based protocol and provides a simple implementation of quenching. Client libraries are available for C, Java, Python, TCL, Common and Emacs Lisp, and Smalltalk. The initial design criteria targeted the implementation at servicing desktop notification service clients in a LAN environment, from which a scale of around a thousand concurrent clients each with around ten subscriptions was determined. Our chief assumption was that changes in subscriptions would be orders of magnitude less frequent than messages, and the resultant system architecture is heavily biased towards rapid evaluation against a relatively static subscription base.

The client API is simple, consisting, for example, of 11 functions in C. Aside from the initial connection, all server interactions are asynchronous, with notification and subscription quench delivery normally handled via callback functions. Each subscription can also specify multi-threaded delivery, using a pool of threads to run the callback function. A polling API is available, but has been used only for the Smalltalk binding where Elvin's use of native threads did not integrate with the runtime system.

```
conn <- elvin_connect(how, hostname, port,
        quench_cb, error_cb, do_poll);

   elvin_disconnect(conn);

   elvin_notify(conn, notification);

sub_id <- elvin_add_subscription(conn,
        sub_expr, notify_cb, num_threads);

   elvin_replace_subscription(conn, sub_id,
        sub_expr);
   elvin_del_subscription(conn, sub_id);

   elvin_free_quench(quench);
   elvin_replace_quench_cb(conn, quench_cb);

sockfd <- elvin_get_socket(conn);
   elvin_dispatch(conn);
   elvin_poll_notify(conn, sub_id, notifica-
        tion);
```

Figure 4: Elvin3 C API summary.

The Elvin3 subscription language is also simple, styled after the C expression syntax, with a handful of predefined functions available for testing attribute existence, (dynamic) type checking, and regular expression matching. Subscription expressions are supplied as strings via the client API and compiled by the server. Multiple subscriptions can be registered using a single connection, and delivered notification messages contain a list of the matching subscriptions whose callback functions are then invoked.

The server architecture is focussed on rapid evaluation and delivery, within the constraints of the service semantics. In keeping with our philosophy of simplicity, the fundamental semantic is "at most once" delivery. After some initial experimentation, and feedback from application programmers, this was extended to guarantee that for a given producer connection, the messages seen by a consumer would retain their order of sending on delivery. Out of order delivery, while enabling lower latency, complicated client programming to an unacceptable degree and the server architecture was modified to support this guarantee.

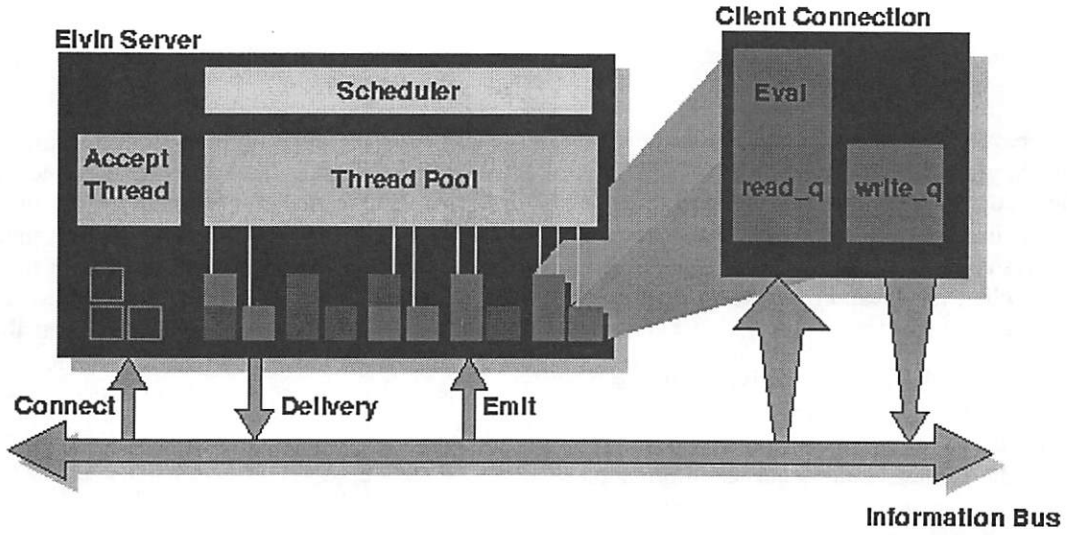

Figure 5: Elvin3 server architecture.

6.1. Performance

One of the chief goals of the Elvin project has been to demonstrate that content-based routing was performant. Existing work in notification services had uniformly chosen to use a channel-based routing approach, which enabled direct use of IP multicast as a routing optimization.

Operational use of Elvin 3 has satisfied this goal: the flexibility engendered by the simple API and subscription language has led to a wide variety of uses with completely satisfactory performance. But quantitative performance measurement is more difficult.

Simple measurements of end-to-end latency show a wide variance, and don't reflect the possible throughput of the server's threaded evaluation engine. It is here that the most difficulty arises: subscriptions are compiled (with some optimization) when registered with the server. The complexity of the registered subscriptions has direct impact on the delivery latency, as does the CPU load on the server host.

Until a comprehensive benchmarking suite for measuring the performance of content-based routing services is developed, it is most useful to measure server performance in terms of "matches per second" where a "match" is a comparison of a message's attribute value against a subscription's requirement.

An Elvin3 server on an AlphaStation 4/255 workstation can perform approximately 200,000 attribute matches per second, and sustain a throughput of 20,000 messages per second (with 50 active subscriptions and a 10% success rate in subscription evaluation).

6.2. User Experience and Issues

After initial testing within the development group, and then wider exposure within our organization, Elvin3 has been deployed across a wide range of external environments. Aside from the usual bug fixes, very few changes have been required, excluding the addition of the polling interface, and the delivery order guarantee discussed above.

Feedback from application programmers was very positive, with the majority of additional comments suggesting extensions to the basic service. Most common amongst these was a nebulous concept unfailingly called "reliability".

Reliability of undirected communications is a difficult concept to define, let alone implement. But Elvin3's asynchronous API and absence of "feedback" to the producer seems to cause a degree of unease in application programmers.

Various degrees of message reliability are possible: from a simple acknowledgement from the server that it has received the producer's message, through to an assertion that all eligible subscribers registered at the time of delivery have confirmed reception of the message. While either of these is simple enough to implement for a single server, we have two primary concerns: that performance would suffer significantly given the overhead of managing acknowledgements and possible attempts to resend lost messages, and perhaps more importantly, that once extended beyond a single server, implementation of anything other than the initial server's reception of the message seems infeasible.

The second most common request is for security (yet another nebulous concept). This was not unexpected, and our progress on a solution is discussed later.

The quench facility in Elvin3 is primitive: the server simply sends a string containing the or-ed subscription strings of all registered consumers. The Python language mapping comes with a set of classes that encapsulate this string, and provide some higher-level manipulation. These classes have been used by a few applications, and this functionality will be merged into the standard API.

Examining the use of Elvin, there are a significant number of applications classified as "one-shot producers": the end result of the application is the emission of a single message. Obviously, the overhead of establishing a TCP connection (and the resultant resources within the server process) is significant compared to sending a single message. However, TCP's reliability is ubiquitous and it is not clear that an alternative reliable protocol would have substantially lower overhead.

Finally, another significant class of applications is what we call *correlators*: subscribers that wait for a specific combination or sequence of messages, possibly within some time constraints, and produce summary messages when their conditions occur. While these applications consume very few resources, they must be long-lived, and ensuring that all the required processes are restarted with the machine, and remain alive is a significant administrative burden. We are investigating a single daemon process that could have registered descriptions of message sequences and timing constraints, and a specification of how to produce the summary message.

7. Elvin 4: Content-based Routing

Elvin3 is a notification service, designed in the tradition of distributed systems, along with name and yellow pages services and an RPC abstraction. But as our understanding of the undirected communications paradigm grew, we began to make the important semantic distinction between a notification service, and a content-based routing infrastructure. Even before starting Elvin3, we had planned to experiment with federation of notification servers, allowing internet-wide subscription and event notification. We now had multiple sites with local Elvin servers wanting to share traffic and the resulting issues of large numbers of users, administrative ownership of servers and their traffic, server redundancy and the like.

In addition, after using notification for communications between desktop applications, it became increasingly apparent that a wealth of activity outside the desktop computer and local area network was useful if made available as notifications.

Both of these directions were instrumental to our developing view of content-based message routing as a fundamental communications paradigm; a similar abstraction to messaging or RPC and critical to the development of disposable computing.

7.1. Experiments with Federation

A single Elvin3 server can handle at most a few thousand concurrent client connections, and while changing to use a connectionless communications protocol would remove the immediate problem, servicing more than a few thousand clients would approach the limits of the host capacity anyway. The real solution is to extend the service beyond a single server, establishing a federation of autonomous servers cooperating to route messages to their consumers.

How will the properties of a federated service have to differ from those of a single server? A single Elvin server provides *universal availability*: a message from a producer is available for delivery to any subscriber (subject to the security scheme described later). But where the Elvin service crosses an enterprise boundary, some filtering of the traffic might be required in a similar fashion to firewalls used at the IP level. While it should be *possible* to receive traffic from any connected server, not all domains will make all messages available.

A single server also provides *immediate visibility*: a new subscription registered by a client is guaranteed to receive a matching message sent as the next packet on a client's connection. It is not feasible to maintain this semantics on a wide-area scale: it would require synchronization of changes to subscription registries. The delayed propagation of both messages and subscriptions mean that this guarantee cannot be maintained for clients connected to different servers.

Finally, the routing of messages between servers introduces the possibility of messages from a single producer using multiple paths to reach a consumer, and hence arriving at a consumer out of order or duplicated. The Elvin3 server is architected to ensure ordering is maintained, and so explicit measures must be taken to carry this over to the service as a whole.

After some experimentation using client programs to filter and forward traffic between Elvin servers, we have settled on two distinct scenarios for federation. They are distinguished mostly by usage requirements, with different trade-offs taken to address these issues in the two contexts.

7.1.1. Local Area Federation

Within an organization, business unit or site, federation usually requires universal availability, and is used as a means of providing reliability, scaling to large numbers of clients or to provide separate administrative authority over a sub-domain. Automatic failover to backup servers, load-sharing ability and flexible configuration are the dominant requirements. Within a local area latency is significant.

If a produced message is effectively multicast to a cluster of Elvin servers, each of which supports a group of subscribers, supporting large numbers of consumers is simply a matter of balancing the consumer connections evenly across a cluster of servers. This mechanism will scale to an almost indefinite number of consumers. Servers have a hand-over facility, allowing a single, advertised server to balance the client load within the cluster. This facility is also used to perform handover of clients for graceful shutdown.

For ease of administration, connections between servers within a local domain are not subject to topology constraints. To ensure messages are not duplicated, regardless of the inter-server links, each message is tagged with sufficient information to detect duplicates which are then discarded. Links between servers are uni-directional, and have optional filters to control message propagation.

7.1.2. Wide Area Federation

Beyond the bounds of an enterprise domain, access to messages is the primary requirement; a communications "backbone" allowing subscription to messages sent from anywhere in the world (or campus, or company) and publication of internal messages for global access.

The primary concern in routing messages beyond a local domain is scalability. Both the traffic volume and the computational effort required to route it must scale to support our quadrillion nodes, many of which will host multiple Elvin clients.

Obviously, simply forwarding all traffic from a local domain onto a global message bus is infeasible. Ideally, only those messages that exactly match the requirements of one or more subscribers, somewhere on the global network, should be sent on. In effect, the backbone should subscribe to a set of messages from a local domain.

However, it should also be possible to prevent both the export and import of classes of messages. An administrator of a domain must be able to apply a filter at the domain boundary, protecting private information from dissemination and restricting the visibility of external events within the domain.

Design of the backbone protocols is still an area of active work. In particular, issues of mobile users and the equivalent of the "Slashdot Effect" [Adl99], where millions of consumers want access to a single message stream, present extreme challenges to the routing infrastructure.

A content-based service has one advantage in scalability; total load on the system is shared across the federation. Each node of the federation only deals with the data once, unlike point-to-point protocols where the originating endpoint must process every request.

7.2. Elvin 4

Elvin4 is an evolution of the Elvin architecture, with refinements across the board from protocol to API. Major changes include

- the introduction of a security mechanism,
- a modular architecture for the underlying marshalling, security and transport protocols,
- automatic server location using SLPv2 [GPVD99],
- better quenching support, and
- an extended subscription language, including support for international strings.

With the facility for multiple protocol stacks supporting the high-level communication, comes the requirement for an interworking protocol to ensure that all Elvin domains can interconnect if required. The combination of XDR[Sri95] marshalling, SSL-3[FKK96] security and TCP/IP transport has been defined as the standard protocol stack, which must be used for to connect to the Elvin backbone.

Despite a more complex internal architecture, we expect significant performance gains from this latest implementation. Careful memory management, a revised threading strategy and better marshalling are targeted at improving the server's bandwidth. Additionally, merging and sharing evaluation graphs [GS94] may lead to signifcant performance increases.

7.2.1. Security

The basic requirements for securing undirected communications are simple: firstly to prevent unauthorized subscribers from receiving messages, and secondly to prevent attackers from "spoofing" messages from a legitimate producer.

A third requirement is introduced by the Elvin quench mechanism. The returned quench messages must not reveal subscriptions for which the producer may not produce matching messages.

In order to retain the loose coupling of content-based routing, we have adopted a mechanism derived from [Pin92], attaching keys transformed by a one-way function to each sent message. Producers retain the raw key, and distribute the transformed key to authorized consumers. When sending a message, the raw, private key is presented, and transformed by the server on arrival. Consumers supply their (already transformed) key when subscribing, and the server compares keys as part subscription evaluation.

Privacy of both the authorization keys and message content can be preserved by encryption of the link between the client library and the server (and between federated servers). Users can specify their preference for security mechanism, authentication and privacy during connection establishment. The use of authentication and privacy is optional, and computationally expensive.

The major problem with this mechanism is that the plaintext of the messages, is exposed to the intermediate servers routing the message to its destinations. Unfortunately, when using the content to perform the routing, this is unavoidable. Of course, it is always possible to encrypt the body of the message prior to transmission if required.

7.2.2. Charging

While we anticipate that most use of Elvin services will be "local" and remain uncharged, the provision of information services and federation of Elvin domains requires a charging model allowing producers to add a premium to the basic transportation costs and backbone routers to allocate forwarding costs to users.

To complicate matters, the cost of routing is not consistent, with complex subscriptions are able to consume significant CPU resources during evaluation. While a simple model of charging by number of bytes is attractive, it does not allow for accurate cost recovery. Additionally, it is not clear that a single charging model will suffice: allocation of the total cost between the producer and consumers of a message could occur in any number of ways, with neither producer only nor consumers only acceptable.

Note that billing is not part of the problem. The Elvin server must simply log the data required for billing which can be processed by a third party.

A simple charging mechanism is provided in Elvin4, but charging in a wide-area Elvin federation remains an open issue.

8. Future Work

Elvin4 is a testbed for our research into the challenges of internet-scale undirected communication using content-based routing. Active work proceeds on federation protocols, the security and charging mechanisms and additional services.

An undirected communication infrastructure would be incomplete without some form of correlation engine (see [LV95]) providing recognition of message patterns. Leveraging previous work in Linda on such recognition, complex correlations can be built from smaller components to embed expert knowledge into the network, for example using a process trellis [FG91].

The availability of access to such a wealth of information from so many devices makes management of an object's relevant context an extremely difficult problem. While the use content-based routing and a correlation service make it relatively simple to create the mechanism for contextualization, the real problem lies in creating policy of sufficient detail for everyday use. People are extraordinarily good at casual awareness and selective focus. The challenge is firstly to simplify mechanisms for defining detailed policy, and secondly to ensure the portability of that policy so it adapts as location changes. We are drawing from related work on awareness and context management in computer supported cooperative work[MKFPFT97, Fit98] to provide objects with the ability to adapt to their surroundings in similar ways to people.

9. Conclusion

Content-based routing is a fundamentally different paradigm for interaction between networked objects. By removing the necessity for producers to direct messages, we gain enormous flexibility in system architecture and scalability over traditional communication systems allowing us to provide an interaction environment for disposable computing.

We are only at the beginning of our exploration of this paradigm but can already see the benefits of decoupling the production, consumption, and dissemination of data between networked components. Undirected communication facilitates systems that are more easily extended, simpler to componentize, and contain a clearer mapping to real world interactions between objects.

Disposable computing requires a revolution in distributed systems; existing paradigms will not scale effec-

tively to support the rapidly changing environment of vast numbers of relevant objects. Undirected messaging overcomes some of the problems of existing systems and provides a viable infrastructure for communication in a quadrillion node network.

And besides, how else will we know where our tomatoes are?

Availability

Elvin is available in both source and binary form under a not-for-commercial-use license. Full documentation, FAQs, additional software and the download itself can be found on the Elvin homepage

http://www.dstc.edu.au/Elvin/

The SLPv2 implementation used in Elvin will also be available independently. See

http://www.dstc.edu.au/Elvin/Sulphur/

Acknowledgements

The work reported in this paper has been funded in part by the Cooperative Research Centre Program through the Department of Industry, Science and Resources of the Commonwealth Government of Australia.

This work has also been supported in part by the United States Defense Advanced Research Projects Agency under grants F30602-96-2-0264 and F30603-94-C-0161 (both administered by the US Air Force through Rome Laboratories). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Bibliography

[Adl99] The Slashdot Effect: *An Analysis of Three Internet Publications*, http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html", 1999.

[CG85] Bill Croft and John Gilmore, *Bootstrap Protocol (BOOTP)*, IETF RFC-951, September 1985.

[DB92] DiBella and Bhandaru, *Pilgrim Event Notifier Version 1.0*, Technical Report, University of Massachusetts, Amherst, November 1992.

[DEFJKS88] DellaFera, Eichin, French, Jedlinsky, Kohl and Sommerfeld, *The Zephyr Notification Service, Proceedings* USENIX Winter 1988, Dallas Texas, pp213-219.

[FG91] Factor and Gelernter, *Software Backplanes, Realtime Data Fusion and the Process Trellis*, Technical Report YALEU/DCS/TR-852, Yale University Department of Computer Science, March 1991.

[FKK96] A. Frier, P. Karlton, and P. Kocher, *The SSL 3.0 Protocol*, Netscape Communications Corp., Nov 18, 1996.

[Fit98] Geraldine Fitzpatrick, **The Locales Framework: understanding and Designing for Cooperative Work.** PhD. Thesis. The University of Queensland, Australia, 1998.

[GAO90] S Gursharan, R Andrews, A Oppenheimer, **Inside AppleTalk**, Addison-Wesley, 1990

[GB92] Gelernter and Bernstein, *Distributed communication via global buffer*, Proceedings ACM Symposium on Principles of Distributed Computing, August 1992, pp10-18.

[GPVD99] Erik Guttman, Charles Perkins, John Veizades, Michael Day, *Service Location Protocol, Version 2*, IETF Internet Draft, work-in-progress, draft-ietf-srvloc-protocol-v2-12, February 1999.

[GS94] J. Gough and G. Smith, *Efficient recognition of events in a distributed system*, Proceedings 18th Australian Computer Science Conference, 1994

[LSWZ97] Leckie, Senjen, Ward and Zhao, *Communication and coordination for intelligent fault diagnosis agents*, Proceedings Eighth IFIP/IEEE International Workshop for Distributed Systems Operations and Management (DSOM'97), Sydney, 21-23 October 1997.

[Low97] Colin Low, *Integrating Communication Services*, IEEE Communications, v35n6, June 1997.

[LV95] David C. Luckham and James Vera, *An event-based architecture definition language*, IEEE Transactions on Software Engineering, 21(9):717-734, September 1995.

[MKFPFT97] Tim Mansfield, Simon Kaplan, Geraldine Fitzpatrick, Ted Phelps, Mark Fitzpatrick, Richard Taylor, *Evolving Orbit: a progress report on building locales*, Proceedings of Group97, ACM Press, Phoenix, AZ, Nov 1997.

[MS91] Chuck McManis and Vipin Samar, Solaris *ONC: Design and Implementation of Transport-Independent RPC*, Sun Microsystems, 1991.

[OMG91] Object Management Group, **Common Object Request Broker: Architecture and Specification**, OMG TC Document 91-12-1, December 1991.

[OMG97] Object Management Group, **Trader Object Services Specification**, OMG TC Document 97-12-23, December 1997.

[OMG98] Object Management Group, **Notification Service: Joint Revised Submission**, OMG TC Document telecom/98-11-01, November 1998.

[Pin92] James Pinakis, *Directed Communication in Linda*, Proceedings 15th Australian Computer Science Conference, January 1992, pp731-743.

[RBM96] Robbert van Renesse, Kenneth P. Birman and Silvano Maffeis, *Horus, a flexible Group Communication System*, Communications of the ACM, April 1996.

[RDR98] Ramduny, Dix and Rodden, *Exploring the Design Space for Notification Servers*, Proceedings CSCW'98, Seattle WA, pp227-235.

[RW97] David S Rosenblum and Alexander L Wolf, A *Design Framework for Internet-Scale Event Observation and Notification*, Proceedings of the Sixth European Software Engineering Conference/ACM SIGSOFT, Fifth Symposium on the Foundations of Software Engineering, Zurich, Switzerland, September 1997

[SA97] Segall and Arnold, *Elvin has left the building: A publish/subscribe notification service with quenching*, Proceedings AUUG97, Brisbane, Australia, September 1997.

[SHMO94] John Shirley, Wei Hu, David Magid and Andy Oram, **Guide to Writing DCE Applications**, O'Reilly and Associates, 2nd Edition, May 1994.

[Ste92] Neal Stephenson,*Snow Crash*, Bantam, 1992.

[Sun88] Sun Microsystems, Inc, *RPC: Remote Procedure Call Protocol Specification, Version 2*,IETF RFC-1057, June 1988.

[Sun99] Sun Microsystems,*Jini Distributed Event Specification*, Technical Report, January 1999.

[Sri95] R. Srinivasan XDR*: External Data Representation Standard*, IETF RFC-1832, August 1995.

[TSS95] Teknekron Software Systems, *Rendezvous Software Bus Programmer's Guide*, 1995.

[Tha99] Thuan L Thai, **Learning DCOM**, O'Reilly and Associates, 1st Edition, April&nsp;1999.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel and Dale Skeen, *The Information Bus: an architecture for extensible distributed systems*, ACM SIGOPS Operating Systems Review, v27 n5, December 1993, pp58-68.

[VGPK97] John Veizades, Erik Guttmann, Charles Perkins, S Kaplan, *Service Location Protocol*, IETF RFC-2165, June 1997.

[Wan95] Want, Schilit, Adams, Gold, Petersen, Goldberg, Ellis and Weiser, *The ParcTab Ubiquitous Computing Experiment*, Xerox PARC Computer Science Laboratory Tech Report CSL-95-1, March 1995.

[Wei91] Weiser, *The Computer for the Twenty-First Century*, Scientific American, September 1991.

[WWWK95] Waldo, Wollrath, Wyant and Kendall, *Events in an RPC Based Distributed System*, SunLabs Technical Report SMLI TR-95-47, November 1995.

Smart Office Spaces

Bora A. Akyol, Matt Fredette, Alden W. Jackson, Rajesh Krishnan,
David Mankins, Craig Partridge, Nicholas Shectman and Gregory D. Troxel.
BBN Technologies
smart-spaces@ir.bbn.com

Abstract

Imagine a world in which every device has an embedded processor and a high-speed wireless link. Any two devices can talk to each other and you link devices together as needed to get your work done.

Devices with embedded processors and wireless links are coming soon. This paper looks at some of the problems we have to overcome to make it possible to link devices together and get work done (rather than cause frustration).

1 Introduction

The advent of embedded processors with built-in wireless transceivers offers a number of chances to revolutionize both communications and computation. We use the term *smart space* to refer to the environment created by a cluster of these wireless-connected processors. At BBN we are investigating the problems of using smart spaces to revolutionize the office environment.

We chose to focus on the office environment for three reasons. First, the overall goal is clear: to enable an office worker to work more effectively with whatever electronic tools are in proximity to him or her. Smart spaces must make work easier (or more straightforward) than current systems or they will not succeed in the office environment.

The second reason for our focus on the office is that it clarifies the role of the Internet. The Internet is one office tool among many; a very important tool, but not the only one. Leslie Lamport once said, "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable," [Lamport]. Businesses spend a lot of money to try to avoid this kind of scenario. At the same time, the profusion of devices in a smart space means the likelihood that some device will fail is high.

The Internet will not always be accessible to a smart space. For example, even users of today's low speed cellular phone networks periodically suffer poor signal quality. So our goal is to create smart spaces that can capitalize on the Internet's resources, when available, but still allow users to work productively when the Internet is not available. A corollary is that Internet access should not be required to complete a purely local transaction.

The third reason to focus on the office is that, being office workers ourselves, we can try to live in the smart spaces we build. Living a vision is a very effective way to find mistakes and identify unexpected benefits.

In the rest of this paper we present what we believe are the key challenges to incorporating smart spaces into the office and then discuss potential solutions that we are exploring to two of those challenges.

2 What is a Smart Space?

Before examining the challenges to creating a smart space in an office, discussion of what constitutes a smart space is worthwhile.

In our vision, every device such as a display, a mouse, a keyboard or a disk drive, contains its own embedded processor and is wireless capable. In such a world, assembling a computing environment should be a matter of placing the necessary parts within wireless signaling distance of each other. Integration of the devices into a computing environment should be transparent to the user.

In an office environment, there is also a premium on minimizing the work necessary to achieve a result. For

instance, we believe that assembling a computing environment in an office is a matter of collecting the devices necessary to perform the current task and we should seek to minimize the number of devices necessary to achieve any one task. For instance, in our view, displaying a file (to be read, or projected in front of an audience) should require the presence of just a display device, a storage device containing the file, and some interface (perhaps a button already on the display device) to enable scrolling forward and backwards in the document. In a wilder example, a confident touch typist should be able to edit a file if she has access to a keyboard and the disk the file is on.

3 Challenges

We see four broad challenges to creating effective smart spaces in an office environment. All four challenges involve eliminating dependencies that hinder flexibility. In a smart space, with a potentially large number of components, even a modest failure rate can lead to a system where the inability to do work is chronic. One way to avoid this problem is to reduce failure rates (which is often hard). Another approach, which we are pursuing, is to vigorously avoid creating dependencies.

The four challenges (all forms of dependencies) that concern us are:

Power dependencies: Power is obviously a pressing concern in smart spaces. With dozens of devices cooperating, the likelihood (with today's technology) that one of them will have a battery running low or will need to be plugged in, is high. Another way to think of this problem is that fully untethered dynamic computing requires us to get rid of two wires: the communications wire and the power wire. Embedded wireless gets rid of the communications wire. We still need to worry about power.

Network dependencies: A lot of problems in smart spaces get easier if you assume that a smart device is constantly attached to the wider Internet. In particular, it is easy to assume that smart devices are configured by means of a network device configuration protocol such as the Dynamic Host Configuration Protocol (DHCP). Furthermore, if a device lacks some piece of information (an applet, some data, or a name-to-address binding) it is often convenient to assume that this information can be retrieved from some repository. Authentication of a user or a device may require an authentication certificate to be downloaded from a certificate authority.

But there are a number of environments where devices may not be connected to the Internet, or may have only intermittent connectivity. In our view, it is unacceptable to design a system that requires Internet connectivity to function correctly.

Peripheral dependencies: It is very easy to build in dependencies on certain types of peripherals. The most obvious example is the keyboard. Many applications unnecessarily assume the user has a keyboard. Any input device (mouse, voice input, or keyboard) should suffice, especially if the user wants to do something simple like deleting a slide from a presentation.

Application dependencies: The major reason many people haul PCs around is that the PC contains the code for all the applications they use. If we are really going to make smart spaces work, we need to solve the problem of the tight coupling between applications and the files they produce or maintain.

An example may help to illustrate this point. Suppose you are going to give a presentation. In the ultimate smart space, all that should be required is a self-reading diskette cartridge that contains a copy of the presentation, a projection device such as a flat screen or projector and some mechanism, such as a button, to advance the slides. But in today's world, this configuration isn't sufficient. You'll also need the software that generated the presentation and knows how to read and display the presentation file. We think that's undesirable: it creates a situation in which it appears to the user (who can see the diskette, projector and button) that she has everything necessary to give the presentation, but in fact it is not possible.

4 Solutions

Enabling a smart office space that empowers the people in that space to work more efficiently requires that we innovate to eliminate the dependencies on power and network connections, and the dependencies on particular peripherals and applications. In this section, we present work at BBN on solutions to power and network connection issues, and sketch some thoughts about how (or how not) to address peripheral and application dependencies.

4.1 Eliminating Power Dependencies

Our vision for a smart office space is truly tether-less. A truly tether-less environment is free from wires not only for communication but also for power.

The idea of wireless transmission of power has been around since Tesla [Brown, McSpadden]. Modern applications that have been proposed range from beaming power by microwave to aircraft to beaming solar energy from stations in outer space to the ground. Indeed one could argue that almost all wireless communications are specific cases of wireless power transmission. Several applications of this technology ranging from electric power transmission to microwave beam powered aircraft have been publicly demonstrated. However the applications considered so far have been on a grand scale at power levels of 500 watts or higher. Proposals also have been made to build generating stations, using photovoltaics and other technologies, on the moon, on nearby asteroids or satellites, from where the power would be beamed down to earth.

At the other end of the spectrum, very low power and extremely short-range wireless powered devices are already in use in medicine. In certain surgical implants it is considered risky or undesirable to include batteries – an external battery carried by the patient is used instead to power the implant wirelessly. Contactless transfer of power to isolated integrated circuits by inductive means also has been proposed [Selvidge].

With increasing power efficiencies of semiconductor computing and peripheral devices, there is an incentive to power devices directly using radio frequencies (RF) which are currently being used for communications. Examples could be electronic product labels and price tags in supermarkets where battery-free operation would be ideal or an embedded temperature sensor in a smart office. Battery-free operation is very attractive considering that there would be fewer environmental side effects.

However, as appealing as the idea of eliminating the battery sounds at first, several applications require disconnected operation both from a wired or wireless power source as well as from the data network. We also have to be careful not to replace a dependence on batteries with a dependence on irradiation. It is more appealing to seek a hybrid approach that both provides power and also recharges an internal battery through wireless means. For example, consider that solar powered watches almost always have an internal rechargeable battery.

4.1.1 Different Approaches

Wireless recharging devices do not currently exist. We are exploring two ways to create them:

The "Diffused Glow Interior Lighting": Buildings will continue to have a wired infrastructure both for power and communications. Addition of wireless power transmitters can be done at low cost.

In this approach, buildings are equipped with power transmitters to power or recharge smart devices. The transmitters omni-directionally transmit diffuse microwave radiation. Devices to be powered or recharged by wireless means have an attached rectenna to intercept the radiated power and convert it to DC.

The "Drying Lamp" or "Microwave Oven" : Bulk recharging of wireless devices is an interesting application worth considering. Devices suitably equipped with rectennas can be recharged en masse by being dumped on to a table and irradiated with microwaves from a projector (drying lamp). Or the irradiation apparatus can be shielded during operation (the microwave oven), permitting higher power densities and quicker recharging. The devices to be recharged this way either need to have their electronics shielded or they can be switched off during recharging.

The implications of wireless powering and recharging are profound. We believe that this technology has a wide range of applications and can revolutionize the way in which we think about and use wireless devices. Some questions remain to be answered regarding the safety and commercial feasibility of wireless power delivery. At the same time, the basic pieces of technology required are already available today. We are working presently to put that technology together into effective demonstrations.

4.2 Eliminating Network Dependencies

A smart device must be able to communicate with other smart devices within its vicinity. This communication should not depend on whether connectivity to the broader Internet is available. Exactly how well a device will function depends, in large part, on the device's purpose. Local operations, such as file transfer or remote

login, between devices in a space should work. Remote operations, such as emailing the home office or resynchronizing with an office calendar program, may not work, or may require that a request be logged, to be fulfilled when Internet connectivity is available again.

Smart devices that communicate with each other without the need for external configuration are referred to as an ad-hoc network (AN). In order to enable ad-hoc networking of smart devices, the wireless channel that these devices share must support a discovery mode and must allow for arbitration of resources. Once the smart devices discover each other, they should be able to communicate without external support. At the same time, if one of the devices has Internet connectivity, Internet access should be available to the entire AN, without requiring the AN to reconfigure itself.

A major challenge is self-organization – enabling the smart devices to form a useful ad-hoc network quickly. It should take very little human input and allow the configuration effort to scale as the number of devices or spaces or both grow large. The mechanisms to create the ad-hoc environment should also be parsimonious in their use of bandwidth. We want to use most of the network's bandwidth to do work, not manage the network.

4.2.1 Security and POKI

Security usually requires the labor-intensive process of integrating a new device into a public-key infrastructure (PKI), so it can use public-key authentication.

Current and proposed public-key infrastructure such as X.509v3 [Housley] and SPKI [Lampson] focus on achieving high assurance of name-key bindings while accepting moderate administrative costs for activities such as key transport to certificate authorities, and the effort to verify identities. While there will always be scenarios which require these high-assurance PKIs, we believe Smart Spaces can not rely on them because the cost of manually configuring every smart device outweighs the need for high assurance. Furthermore, we believe that productive work among local smart space devices must be possible without a access to certificate hierarchies outside the space. For example, using a PKI that requires Internet or Intranet access to verify a name-key binding is not consistent with our design philosophy.

We believe that a new PKI can be designed that

1. needs no pre-configuration,
2. exchanges key information when learning about device capabilities,
3. allows varying degrees of assurance about keys bound to a name, and
4. allows automatic update of assurance estimates about keys.

First, let us look at an example of why increasing assurance will work for Smart Spaces, by investigating what happens when the residents of a neighborhood meet a new neighbor. At first the new neighbor is outwardly accepted at face value, but a resident does not really have any assurance that the new neighbor is trustworthy or even owns the house. Over time, as the putative new neighbor continues to act like the new neighbor, the estimate of assurance goes up.

In neighborhoods, people also talk about each other, and especially about a new neighbor. A new neighbor is likely to be described (and in the digital world, this would include a public key) to other residents who have yet to meet him. Thus, by the time the n^{th} old resident meets the new neighbor, their estimate of assurance of the new person's identity (the public key belonging to the new resident is K) is already fairly high.

While some PKI systems such as PGP [Callas] and X.509 with cross-certification do this with mechanisms likely to involve humans, we describe a mechanism that functions without human intervention, and so makes good sense for Smart Spaces. Our Probabilistic, Opportunistic Key Infrastructure (POKI), defines how nodes go about creating a security infrastructure simply by communicating with one another.

When two devices that speak POKI meet, they both exchange keys, and then gossip. The gossip they exchange is the names of other devices that they know, the keys that they associate with those names, and their estimate of assurance that each key is correct. In addition, they may also choose to gossip about what *other* nodes' estimates of assurance are in the different keys.

After the two nodes have exchanged all of the information (and performed the transaction the connection was started for), they execute their algorithms for updating their beliefs about names, keys, and probabilities.

With POKI, a large number of small devices can gain a moderate level of assurance about the keys in the Smart Space without pre-configuration. The gossip can occur when devices in the space inquire about each other's ca-

pabilities. Assurances that started out a low levels can increase over time without any human intervention.

4.2.2 Directories

Directories are also a good example of the need to eliminate network dependencies. Today, directories are a vital piece of network and systems infrastructures. We rely on a networked directory to translate domain names to network addresses. We rely on web client and operating system directories to tell us what application is best suited to opening a file. Yet directories are often problematic in an ad-hoc space.

The problem is best explained by examples. Suppose Cheryl and David walk into a conference room. Cheryl has a copy of a document that David needs and both PCs are part of the local ad-hoc network, but are not currently connected to the Internet. Cheryl should be able to tell David the name of her PC and he should be able to FTP a document from it. That means that the traditional domain name system lookup, which depends on the Internet's distributed domain name system, will have to be bypassed and use some local directory service to map the name of Cheryl's machine into a local address.

Now consider the problem of displaying a document from a web browsing device. A web browser typically contains an internal directory mapping file names to applications that can display them. In an ad-hoc space, the applications will generally not reside on the browsing device, but will be distributed on various storage devices, which may or may not be carried by the user of the browser. How does the browser find the right application (of the right version, that runs on the browser's processor) in the local ad hoc network?

Note that a straightforward approach, namely registering with a registry when one joins a smart space, doesn't work well. First, the assumption of a registry device introduces a dependency on the registry, both that it exists and is functioning in the space. We are just trading one dependency for another. Second, the volume of information that needs to be registered could be quite high (e.g., a list of all executable applications on a hard disk) and changing rapidly (that hard disk may have been in the pocket of someone who just walked past your conference room). Third, the bandwidth available to some of the smart devices may be low, hence making access to a registry that covers a large number of items expensive.

We suspect that the registry may have to be fully distributed. That is, all interested devices keep track of the part of the registry they care about, and devices advertise their properties to everyone. Some balance of pushing data, and querying for data when the data is needed, will have to be found. Alternatively, a service resolution protocol might be useful. A service resolution protocol multicasts requests for particular services, as needed. The advantage of such a protocol is that it is demand driven. No one needs to keep track of temperature sensors if no application is using one.

4.3 Eliminating Peripheral Dependencies

Dependencies on peripherals are often created by the programmers of applications that do not create alternate means of inputing commands or data to change the way an application behaves. For instance, in our view, if a user has access to a display and an input device, the user should be able to edit a document on the user's disk. But in today's world it is all too likely that the user's document editor requires both a mouse and a keyboard (even though one or the other is sufficient).

In order to enable smart devices to overcome peripheral dependencies, a capability description language may be appropriate. Such a language would allow the smart devices to query their peers to resolve their capabilities. So, an application might query an input device about whether it could emulate a mouse. And the slide projector in a conference room could advertise its resolution and supported scan rates.

But there's a challenge hidden in the idea of a capability language: how many different types of devices are there, and how do we represent them? For instance, a mouse can come with one to three buttons. An application will have to behave differently, depending on the number of buttons. But the last thing we want is a world where an application refuses to work because the user has a one button mouse and the application expects a three button mouse.

This problem was recognized many years ago by the ARPANET pioneers and dubbed the *m-by-n* problem. The basic idea is that if there are m applications that know the details of n different devices, then cost of adding the $n+1$st device to the mix is often prohibitive, because it requires updating the m applications. The trick is to actually have as few distinct types of devices as possible. So, for example, we could classify keyboards, pointers and mice, as input devices, capable of giving us a single one-button signal, and require every application

to work this device. We can then allow optional negotiation of additional capabilities such as "can you send character codes?" and "do you have more than one button?"[1].

Developing these kinds of standards requires tremendous community effort and a great willingness to radically simplify. But it has tremendous advantages.

4.4 Eliminating Application Dependencies

As we noted above, many files are closely related to particular applications. You often cannot effectively open or manipulate a file except with the application that created it.

There are obvious solutions to eliminating application dependencies, all of which are problematic. One solution is to combine the file with its application. So every PowerPoint[2] document also includes a (platform independent) version of PowerPoint viewer. This result consumes disk space with many useless replications of our application. A slightly better approach is to put one copy of the PowerPoint viewer on any device that contains a PowerPoint file, but that still is intensive.

The UNIX solution of fairly generic text files softens the problem but does not solve it. We could, for instance, edit the source files for this paper with a range of applications (*emacs*, *vi*, *sed* and *awk* are obvious examples). But trying to format the source using *groff* instead of LaTeX would be problematic on a deadline.

New ideas are needed in order to make advances in this area.

5 Conclusion

Smart Spaces is a very rich research topic, filled with interesting problems. Our particular focus on the office environment has presented us with an inviting set of challenges to solve. We believe we have two exciting solutions: a method for wireless power delivery and a mechanism for key management in ad-hoc environments. Although we continue to be challenged by peripheral and application dependencies, we have identified problems in the obvious solutions and look forward to working on these problems more in the future.

[1] This solution is precisely the one the ARPANET designers choose for supporting terminal types over telnet

[2] PowerPoint is a trademark of the Microsoft Corporation.

References

[Brown] W. C. Brown, "The history of wireless power transmission," *Solar Energy,* Vol. 56, No. 1, pp. 3-21, 1996.

[Callas] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP message format," Request for Comments (Proposed Standard) 2440, Internet Engineering Task Force, November 1998.

[Housley] R. Housley, W. Ford, T. Polk, and D. Solo, "Internet X.509 public key infrastructure certificate and CRL profile," Internet Draft, Internet Engineering Task Force, September, 1998, Work in progress,
`http://www.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt`

[Lamport] L. Lamport, Private communication to src-t bulletin board at the Systems Research Center of Digital Equipment Corporation on May 28, 1987.

[Lampson] B. Lampson, T. Ylonen, R. Rivest, W. Frantz, C. Ellison, and B. Thomas, "Simple public key certificate," Internet Draft, Internet Engineering Task Force, March 1998, Work in progress,
`http://www.ietf.org/internet-drafts/draft-ietf-spki-cert-structure-05.txt`

[McSpadden] James McSpadden, "Collection of Power from Space References,"
`http://www.tsgc.utexas.edu/tsgc/power/general/refs.html`

[Selvidge] Charles Selvidge, Adam Malamy, and Lance Glasser, "Power and communication techniques for physically isolated integrated circuits,"
Proceedings of the Stanford Conference on Advanced Research in VLSI, pp. 231–247, May 1987.

AirJava: Networking for Smart Spaces

Kevin L. Mills
Information Technology Laboratory
National Institute of Standards and Technology

Abstract

Increasingly people work and live on the move. To support this mobile lifestyle, especially as work becomes more intensely information-based, companies are producing various portable and embedded information devices. Concurrently, some interesting pico-cellular wireless technologies promise to outfit these portable and embedded devices with high bandwidth, localized, wireless communication capabilities that can also reach the globally wired Internet. An impressionist painting emerges of small, specialized devices roaming among islands of wireless connectivity within a global ocean of wired networks. Each wireless island becomes a "Smart Space", where available services and embedded devices can be discovered, accessed, interconnected with portable devices carried onto the island, and then the combination of imported and native devices can be exploited to support the information needs of the current island inhabitants. In this paper, I outline three specific human-information interaction challenges that the research community must address in order to reap the benefits of specialized information devices within Smart Spaces. Before these research challenges can be adequately addressed, the research community must have some Smart Spaces with which to experiment. I describe ***AirJava***, which combines Java Jini with pico-cellular wireless technology to empower small devices to discover each other, to exchange programs, and to interact. While a technology like ***AirJava*** should emerge in the next five years, I propose a means of building ***AirJava*** adapters today so researchers can begin experimenting with Smart Spaces.

1. Introduction

Increasingly people work and live on the move. To support this mobile lifestyle, especially as work becomes more intensely information-based, companies are producing various portable and embedded information devices. Consider for example, portable digital assistants[1], cellular telephones[2], the CrossPad[3], the InfoPen[4], active badges[5], intelligent buttons[6], and the Internet Car[7]. Concurrently, some interesting wireless technologies, including Bluetooth[8], IrDA[9], and HomeRF[10], promise to outfit portable and embedded devices with high bandwidth, localized wireless communication capabilities that can also reach the globally wired Internet. An impressionist painting emerges of small, specialized devices roaming among islands of wireless connectivity within a global ocean of wired networks. Each wireless island becomes a "Smart Space", where available services and embedded devices can be discovered, accessed, interconnected with portable devices carried onto the island, and then the combination of imported and native devices can be exploited to support the information needs of the current island inhabitants. This painting suggests some wonderful potential outcomes, once a number of research challenges have been mastered. In this paper, I outline three specific human-information interaction (HII) challenges that the research community must address in order to reap the benefits of specialized information devices within Smart Spaces. Before these research challenges can be adequately addressed, the research community must have some Smart Spaces with which to experiment. I describe ***AirJava***, which combines Java Jini[11] with pico-cellular wireless technology to empower small devices to discover each other, to exchange programs, and to interact. While a technology like ***AirJava*** should emerge in the next five years, I propose a means of building ***AirJava*** adapters so researchers can begin experimenting with Smart Spaces today. ***AirJava*** adapters have been proposed to DARPA as one component of a Smart Spaces test bed.

2. Three Challenges for Smart-Spaces Researchers

I predict that within five years specialized devices can be worn or toted into Smart Spaces, and once there can discover and interact with embedded devices at the basic level of exchanging data and programs. If I am correct, then researchers should be tackling many issues today in order to exploit this capability when the time comes. From among the myriad issues to consider, I selected three to discuss here.

2.1 Removing the Tyranny of an Interface Per Device

As many specialized devices become available, human-information interfaces can be distributed across devices and interaction modes. I call such interfaces poly-device, poly-modal (PDPM) interfaces. Depending upon application requirements, user preferences, and knowledge about human awareness, about specific tasks, and about the type of information being conveyed, tomorrow's PDPM interfaces must coordinate interactions across devices and among modalities. Surely, some sort of distributed coordination bus will be needed to exchange interaction events. In addition, a model of interaction events will be needed, as well as rules for mapping between the interaction event model and mode-specific interactions. Given a fluid set of devices available in any particular Smart Space, software mechanisms must support the dynamic composition of interfaces from among software components. Not only must composition be supported, but also rules for instantiating the optimal PDDM interface for specific tasks, given available devices and modalities. Naming and identification will be a key issue, along with authentication and access control. Since information and interaction events will fly through the air, privacy will also be important. Other issues will arise regarding shared access to devices within a Smart Space.

Some researchers are already looking into a few of these concerns. For example, the DISCIPLE[12] project at Rutgers CAIP has integrated into a single desktop interface a range of multi-modal technologies, including gaze and gesture tracking, voice recognition and speech synthesis, along with the typical display, mouse and keyboard. Similarly, the Virtue[13] project at UIUC has developed and is experimenting with a multi-modal interface for immersive virtual environments. At CMU, the Experience-on-Demand[14] project has enabled individuals to collect multimedia records of their experiences and has begun to examine how such individual experiences can be collected into a searchable database. Several collaboration technology projects are developing and evaluating multi-party distributed event buses.[12,15,16] In addition, a few projects have begun to consider the implications of multimedia, cross-device drag-and-drop[15,17].

2.2 Moving Information for People.

Some researchers believe that we will carry all of our information with us as the miracle of hardware continues to bring us ever-increasing density in disk storage, along with cheaper and faster processors. I don't believe this to be the case because human activities continue to produce information at a prodigious rate. In fact, much of the information we produce is context-dependent. For example, we typically attend meetings to conduct specific tasks. Before, after, and during these meetings we create information. Some of this information we retain personally, while other information is shared among the meeting attendees and others outside of the group. Only a small fraction of this information is our own personal information. Surely, as we move to the next meeting on the same subject we will wish to have information from the last meeting available. I argue that context can often be inferred from a combination of user, location, and task. If so, then why should a user be required to ensure that the right information is available at the right place and time? Can't the information itself take on this responsibility? Imagine active information objects that can move, that can replicate themselves, and that can communicate as a group. Such active information should be able to track the location, state, and trajectory of information users, of object replicas, and of linked objects. In addition, active information objects should be able to plan the movement, replication, and transformation of information to serve the projected needs of its users. Active information objects must also be able to implement consistency, access, and sharing policies among replicated and linked objects.

A combination of commercial and research activities show some promise that a day will soon appear in which active information becomes possible and interesting. Clearly mobile code systems such as Python[18], Aglets[19], AGNI[20], and Java[21] hint at the possibility of distributed object systems that can replicate and move. From the research world, the BARWAN[22] and MASH[15] projects at UCB are developing scalable reliable multicast protocols, beaconing protocols, and transcoding algorithms that distributed objects can use to discover each other, to communicate, and to transform their presentation. In addition, the Networks of Workstations[23] and Active Services[24] projects at UCB promise a processing-capable network infrastructure that can provide a platform for mobile distributed objects to reside within the network and to move or copy themselves toward specific Smart Spaces as users begin to congregate.

2.3 Adapting Information Delivery Using Knowledge of People, Places, and Devices.

At present, networked-based computing works because people carry in their heads a reasonably good model of cyberspace. We know where computers and printers can be found; we know how information can be organized for storage on a disk. We know, but just barely, how to locate, download, configure, and execute various plug-

ins to display information in specific formats or to convert information between formats. In fact, sometimes I think our computers and networks should pay us because we sure do a lot of work for them. Suppose, on the other hand, that our computers and networks had a much better model of the world in which we live. Would it be possible for our computers and networks to help us more than we help them today? I suggest that researchers consider trying to build Inter-Space, a model that crosses the gap between physical and logical space as we perceive it and cyberspace as it exists in our computers and networks. Suppose we could couple sensor data with resource and scene description languages to model within our computers the physical and logical space (maybe physilogical space) that people perceive and understand. If we could, then our software might be able to exploit location, proximity, and visibility of resources to determine where to deliver specific services for us. In addition, our software might be able to adapt information presentation to the specific characteristics of available devices and services. In fact, more generally, if our software has a model of physilogical space that appears reasonably consistent with our own, then we might be able to encode into our computer heuristics similar to those that we now use when reasoning on our own about cyberspace.

Think about it. Sensors of all kinds are becoming cheaper and more capable. These include digital still and video cameras[25,26], digital sensors[27], eye-tracking devices[28], radio-frequency tags[29], and global positioning system chips[30]. In addition, a few projects, such as Dataman[31] at Rutgers and the BARWAN[22] and MASH[15] projects at UCB are beginning to explore location-based networking services. Research projects at UCB[32] and at MIT[33] are also investigating methods to construct geometric models automatically from still pictures and video images.

I discussed only a few among many fruitful topics that can be explored in the context of Smart Spaces. The main concern of the remainder of this paper is to explain how researchers can begin to investigate Smart Spaces today.

3. *AirJava*

Two emerging technologies, pico-cellular wireless and mobile code, promise to provide the networking foundation for Smart Spaces. I envision that within five years vendors will offer portable and embedded devices containing low-cost chips for pico-cellular wireless communications, coupled with virtual machine interpreters that support the exchange of executable programs among networked devices. The combination of these new technologies promises to revolutionize computing and networking, as we know it today. I propose that we can accelerate this coming revolution by designing and constructing ***AirJava*** adapters that can convert any existing computer-controllable device into a prototype Smart Spaces device. Further, I propose that ***AirJava*** adapters can be used within Smart Spaces test beds, and can be supplied to researchers investigating issues associated with Smart Spaces. ***AirJava*** adapters should also inform commercial developers about design issues surrounding Smart Spaces devices. Here is how ***AirJava*** could be realized, and how Smart Spaces test beds could be created.

3.1 Pico-cellular Wireless Technology

Industry is developing two, competing wireless "transceiver-on-a-chip" technologies that can provide relatively high bandwidth (300 Kbps to 1.2 Mbps) over limited ranges (10 to 50 meters) by exploiting the unlicensed frequency band around 2.4 GHz. Table 1 gives the specifications for one of these technologies, Bluetooth[8], while Table 2 gives the specifications for the other, HomeRF[10].

While these two radio frequency (RF) technologies compete for a similar market, no matter which one prevails, future portable and embedded computing devices will clearly come equipped with some RF technology that will permit relatively high speed communications over a restricted range. Such technology will provide the communications conduit for large numbers of transient devices to begin to talk. But how will these devices discover each other and establish links and what will the devices say to one another? That's where mobile code comes into the picture.

Table 1. Planned Specifications for Bluetooth Version 1.0

- Range 10 Meters in shirt pocket or briefcase
- Network Size: 8 devices per pico-net
- Data Rate: 300-400 Kbps
- Frequency: 2.4 GHz and 1600 Hops/sec
- Supports 3 near line-quality voice links
- Optimized for cell phones and mobile devices
- Multi-point to point connections
- Point-to-point TCI/IP support
- Low power standby mode
- Higher transmit power possible
- Based on a working prototype
- More: http://www.bluetooth.com
- Main players: Ericsson, IBM, Intel, Nokia, and Toshiba

Table 2. Planned Specifications for HomeRF Version 1.0

- Range: 50 Meters in home and yard
- Network Size: unlimited
- Data Rate: 1.2 Mbps
- Frequency: 2.4 GHz and 50 Hops/sec
- Supports 6 near line-quality voice links
- Optimized for home voice and data
- Peer-to-peer networking
- Native TCI/IP support
- Low power paging mode
- Lower transmit power possible
- Based on shipping 802.11 and DECT technology
- More: http://www.homerf.org
- Main players: AMD, Ericsson, HP, IBM, Intel, Microsoft, Motorola, National Semiconductor, and more

3.2 Mobile Code

Most people have heard about the Java[21] promise of write-once, run-anywhere software. Java makes this possible by requiring Java-compliant systems to implement an interpreter for a Java Virtual Machine (Java VM[34]). In fact, a number of initiatives are underway to design chips that run the Java VM in hardware. Additionally, Sun and IBM have been designing an operating system, JavaOS[35], intended to provide operating services native to Java programs by implementing a small kernel around a JavaChip[36]. No matter what the outcome of these explorations, it appears possible to implement the Java VM on a chip (Java or otherwise) with fairly substantial memory and reasonable speed. Combining a Java VM-on-a-chip with a RF transceiver-on-a-chip could provide an interesting basis for networking Smart Spaces devices, especially with the advent of Jini[11].

Jini is a Java-based networking technology recently announced by Sun. Jini enables devices, newly added to a network, to discover a lookup service and to deposit there some key information. This information can include a description of the device and its services, along with Java classes that can be used by others to communicate with the device. In addition, Jini enables programs and devices to discover other devices in an area, and then to download Java code that permits communication with the discovered devices. Along with Jini comes extensions to the Java VM to support event distribution among distributed Java VMs, and extensions to Java Remote Method Invocation[37] (RMI) to exploit multicast protocols.

As should now be clear, the ingredients exist to provide Smart Spaces devices with powerful networking functionality in a small, low-power package. Such a package would include a RF transceiver-on-a-chip, a hardware implementation of the Java VM, and enough memory to run the Jini discovery protocols, to hold Java classes for uploading to a Jini lookup service, and to execute Java classes downloaded from a Jini lookup service. My proposed ***AirJava*** adapters would demonstrate the feasibility of these ideas, while at the same time providing a means to prototype tomorrow's Smart Spaces today.

4. A Smart Spaces Test Bed Based on *AirJava* Adapters

The following paragraphs discuss how ***AirJava*** networking for Smart Spaces can be brought to life. Three steps are needed: design of the adapter, production of some prototypes, and demonstration of a Smart Space.

4.1 Design *AirJava* Adapter.

Design of an ***AirJava*** adapter must cover both hardware and software. Hardware for a prototype ***AirJava*** adapter must provide a reasonably capable CPU, a significant amount of flash memory, a modest amount of DRAM, a serial port, a parallel port, a universal serial bus port, and the ability to accept PCMCIA interface cards, including either a RF or IR interface. The packaging must be reasonably compact, and power must be provided with batteries. To support development, the adapter should also accept PCMCIA interfaces for a secondary storage device, such as floppy disk. Software for the prototype ***AirJava*** adapter must provide an operating system or run-time environment capable of executing a Java Virtual Machine and running Jini.

Two approaches to design appear feasible. The conservative approach would base the ***AirJava*** adapter on a small, portable computer, such as the Toshiba Libretto[38]. The Libretto provides a fully functional PC-like device in a reasonably compact form, powered with batteries. The Libretto can accept up to two PCMCIA cards. The Libretto can run Windows 95, Windows 98, or Windows NT, and thus can easily execute a Java

Virtual Machine and Jini. On the downside, the Libretto is a bit heavy (just under 2 pounds) and has a short battery life (around 2 hours). As an alternative to the Libretto, we could consider some of the newer WindowsCE[39] devices beginning to appear on the market. It is unclear whether WindowsCE will support the Java Virtual Machine; however, such devices are generally smaller and have longer battery life that computers in the Libretto class.

A less conservative approach would base the ***AirJava*** adapter on GUMPS[40], a GloMo Universal Module Packaging System developed for DARPA by USC-ISI. GUMPS provides a compact brick that includes a rigid-flex circuit board, a PCMCIA-format CPU, and up to seven PCMCIA cards. Four batteries, which can yield up to 10 hours of operation, power GUMPS. The GUMPS project also envisions a mini-brick (about the size of a pack of cigarettes) that would include a CPU plus up to five PCMCIA cards. The mini-brick would be powered on only two batteries, but an operating life of 20 hours is anticipated due to better power management. While the original plan for GUMPS called for a 33 MHz 486 CPU, as of 1998 a prototype GUMPS unit included a 100 MHz 486DX4 CPU. The 1998 GUMPS prototype also included 150 Mbytes of flash storage. Producing a design based on a GUMPS-like architecture would enable us to consider a range of options for the CPU, including lower power RISC machines and JavaChips. We would also be free to consider software other than Windows. Options might include a free Unix variant, a real-time operating system, or JavaOS. Taking this approach would probably benefit from some form of collaboration with USC-ISI.

4.2 Produce Prototype *AirJava* Adapters.

Once a design has been selected and verified, probably by implementing at least two working prototype ***AirJava*** adapters, at least another ten adapters must be produced to support the planned Smart Spaces demonstration. In addition, we need to provide a cookbook of instructions so that others can produce their own ***AirJava*** adapters.

4.3 Demonstrate a Smart Space.

To show that the ***AirJava*** adapters work as expected, and to illustrate the concepts underlying networking for Smart Spaces, I propose to demonstrate a small, Smart Spaces environment. Our demonstration will include one Jini Server (connected to the global Internet as well as to a wireless cell), two portable computers, and a range of other devices. The range of other devices under consideration include a video projector, a videocassette recorder (VCR), a digital camera, a large-screen display, audio-visual equipment (such as found in the MASH room produced for DARPA by UCB), cell phones, the Cross Pad, wearable computer systems, and personal digital assistants. We need to demonstrate that devices can be brought into a wireless cell, register with a lookup service, and be discovered by other devices and by individuals. We also need to show that appropriate devices can download interface code for registered devices and then use that code to communicate with and control the registered devices. A canonical demonstration might have a person bring a video projector and VCR into a room, plug them into electrical outlets, and then leave. No other wires would be required. The devices would discover the lookup service and upload interface code. Later, a second person can enter the room with a portable computer. Using only the portable computer, the person will discover the interface code for the projector and VCR, download that code, pop in a video tape, remotely configure the VCR output to input to the projector, and then remotely control the playing of the tape, all from the portable computer. This will be accomplished without connecting the computer, VCR, and projector with cables. While the tape is playing, a third person can enter the room with a portable computer. This portable computer can also discover and download the interface code for the VCR and projector. Now, the new person can share control of the VCR with the existing person. Either person can pause, fast forward, play, and rewind the tape. Again, no communication wires. Whatever the specific demonstration chosen, the existence of Smart Spaces test beds will free the research community to consider how Smart Spaces can be integrated with wired networks.

5. Smart Spaces Islands in the Internet Ocean

Given my view of Smart Spaces as islands of high-speed wireless connectivity in an ocean of wired Internet connectivity, interesting issues need to be resolved regarding the relationship between the islands and the ocean. What is the relationship between the local Jini lookup service and emerging global-scale lookup services for the Internet? How can multicast addresses be managed so that some have local scope, while others have global scope? What is an appropriate naming scheme that overarches the Internet and local wireless pico-cell networks? How should security services and protocols be designed to support roving wireless devices that reconnect to the Internet for information access? The point of these questions is to illustrate that as the number of wireless devices increases, the urgency of

research addressing the integration of wired and wireless networking will also increase. The research community cannot afford to wait for the problem to exist; otherwise, the chaos and confusion that will result from various commercial vendors adopting ad hoc solutions will inhibit market growth in wireless devices.

6. Conclusions

In this paper, I have advocated that researchers start today to address both the networking challenges and the human-information interaction issues of Smart Spaces. To permit work to begin soon, I have suggested one means to prototype Smart Spaces by integrating some emerging pico-cellular wireless and mobile code technology into the type of Smart Spaces interface that should be widely available in the next five years. Along the way, I have highlighted some early research that lights the path toward Smart Spaces.

7. References

1. http://www.pdapage.com/
2. http://www.portableconcepts.com/
3. http://www.crosspad.com/
4. http://www.symbol.com/data/std00055.htm
5. http://www.ics.agh.edu.pl/ABng/
6. http://www.ibutton.com/
7. http://www.daimler-benz.com/ind_gfnav_e.html?/research/text/70430_e.htm
8. http://www.bluetooth.com/default.asp
9. http://www.irda.org/index.asp
10. http://www.homerf.org/tech/
11. http://sun.com/jini/
12. http://www.caip.rutgers.edu/multimedia/groupware/darpa.html
13. http://vibes.cs.uiuc.edu/Project/VR/Virtue/VirtueOverview.htm
14. http://informedia.cs.cmu.edu/
15. http://mash.cs.berkeley.edu/
16. http://www.dstc.edu.au/wOrlds/
17. http://www.maya.com/visage/link/
18. http://ftp.python.org/
19. http://aglets.trl.ibm.co.jp/
20. http://snad.ncsl.nist.gov/antd-staff/mranga/agni/
21. http://www.sun.com/java/
22. http://daedalus.cs.berkeley.edu/
23. http://now.cs.berkeley.edu/
24. http://ninja.cs.berkeley.edu/
25. http://www.conde.com/cameras/index.html
26. http://www.videocamera.com/
27. http://www.tpico.com/
28. http://www.visioncs.com/eyet.htm
29. http://www.idsystems-dialoc.com/
30. http://www.penton.com/ed/Pages/magpages/july0698/ti/0706ti3.htm
31. http://athos.rutgers.edu/dataman/
32. http://www-video.eecs.berkeley.edu/~nlachang/MVR/
33. http://graphics.lcs.mit.edu/city/city.html
34. http://java.sun.com/docs/books/vmspec/
35. http://java.sun.com/products/javaos/
36. http://www.sun.com/microelectronics/picoJava/index.html
37. http://java.sun.com/products/jdk/rmi/
38. http://www.csd.toshiba.com/cgi-bin/WebObjects/Toshiba.woa/-/ProductFamily.wo?productTypeId=1&productFamilyId=3
39. http://www.microsoft.com/windowsce/default.asp
40. http://www.isi.edu/asd/gumps/

Bringing the Internet to All Electronic Devices

Michael Howard
Chief Architect, emWare Inc.
Christopher S. Sontag
CTO and co-founder, emWare Inc.

Abstract

In order to develop appropriate solutions for embedded device networking, we must understand the benefits offered to the end user of the device as well as the costs involved with delivering a solution. As proponents of networking technology, it is tempting to overestimate the perceived value of connectivity, while at the same time overlook the hidden costs in implementation. It is important to remember that companies adopting technology for embedded device networking are manufacturers who tend to be very conservative due to narrow margins and fierce competition.
A prerequisite to wide adoption of this networking technology is a convincing case for a strong return on investment. Therefor, all optimism must temporarily be put aside for a critical cost/benefits analysis that will provide criteria for judging the suitability of proposed solutions.

Since there are so many embedded applications, it is not surprising that quite a few applications are 'no-brainers' which require little deliberation as to value or implementation method. Administration of networking hardware, for example, is increasingly accomplished through Internet-based device interfaces.

Internet standards and associated technologies provide a remarkable set of opportunities for enhancing the value of existing and new embedded products. These Internet technologies have been available to millions of large, 32- and 64-bit systems for some time. The key challenge we have undertaken is to establish a distributed device-networking platform that provides appropriate solutions for all embedded devices. Our goal is to make connectivity practical even for systems that do not have convenient networking already available and are severely constrained by economic pressures.

1.0 The Problem

Networking embedded devices is a tremendous challenge because the size of devices, the kind of processors, the type of network, and the information to be extracted from devices vary greatly. The device itself can't bear the majority of the burden of being networked. The device can't bear the majority of the overhead to provide security and it can't bear the burden of a too-high price point. And, in some cases, there may not be value in devoting resources to enable a device to do much more that it already does – due to the simple economics of low cost electronic devices. Technology must adapt to the reality of low-cost devices, not the other way around. Current solutions are too specific to either a certain programming language, physical network wire, transport communications protocol, or operating system. For example, http has become the solution for the Internet because it's generic. We can learn from this model. Many aspects of the problems are the same—the variety of interfaces, the range of processor capabilities, the need for real-time state communication and connection-based control, and the batch data collection and datagram dissemination.

In order to develop appropriate solutions for embedded device networking, we must understand the benefits offered to the end user of the device and the costs involved with delivering a solution. As proponents of networking technology, it is tempting to overestimate the perceived value of connectivity, while at the same time overlook the hidden costs in implementation. It is important to remember that companies adopting technology for embedded device networking are manufacturers who tend to be very conservative due to narrow margins and fierce competition.
A prerequisite to wide adoption of this networking technology is a convincing case for a strong return on investment. Therefore, all optimism must temporarily be put aside for a critical cost/benefits analysis that will provide criteria for judging the suitability of proposed solutions.

Since there are so many embedded applications, it is not surprising that quite a few applications are 'no-brainers' which require little deliberation as to value or implementation method. Administration of networking hardware, for example, is increasingly accomplished through HTTP-based device interfaces. Our goal is to make connectivity practical even for systems that do not have convenient networking already available and are severely constrained by economic pressures.

The cost of implementation results not only from the hardware and software used in communication, but also from:

- *The complexity of a system that must be understood by the applications engineer.* In situations where no device networking is currently in place, a great deal of education must happen in order to integrate communications into an application. The impact on the application in terms for timing, power consumption, noise, and memory consumption must be understood and dealt with.

- *Education of the end user in terms of marketing, documentation and customer support.* The person who stands to benefit from the networking of embedded devices is very likely unaware of the advantages a networked system has over an isolated one. The value added to an application needs to be communicated in order to influence purchasing decisions. Products that might have been closed to configuration and monitoring must now be explained in greater detail in order to enable more intelligent use. Customer support must effectively deal with a new set of problems in order to insure that this new technology does not prove more trouble than it is worth. The cost to a user is not simply based on dollars, but also on the added time and attention that may be required.

- *Tools required for implementation and production.* A set of non-recurring costs for implementation of networking includes all the development tools and changes to a manufacturing process required by the technology. Packaging and government required process changes may be considerable.

- *Embedded device hardware costs for additional resources.* The addition of communications capability will likely require the addition of hardware, including additional CPU power, memory, communications hardware, connectors, and power supply output.

- *Continued support of interfaces that rely on components and are changing over time.* If an industry standard Web browser is used to implement a device user interface, then the manufacturer of the device is burdened with the need to continually insure that changes to the browsers supported do not break the interface delivered to the users of their product. In addition, the browsers must support components like a Java VM. If the support for a particular language feature is altered in any way, the device interface may need to be updated. There seems to be an inherent incompatibility between the rapid evolution of software components and the life of an embedded system. The manufacturer must predict the cost of any such burden. This cost is particularly troublesome to estimate.

- *Additional development and test time to integrate communications functions.* As development cycles are shortened due to market pressures, the adoption of technology that might, at least temporarily, lengthen the cycles for a particular company may cost that company in terms of missed marketing opportunities.

The product developer who must predict total implementation costs evaluates these costs in terms of recurring costs versus non-recurring costs. The manufacturer of high volume products must pay careful attention to the cost of materials and labor, spreading the engineering costs out until the engineer cost per product is insignificant. The products that are created for smaller markets must recover a larger portion of these costs per unit. In order to facilitate rapid adoption of device networking technologies, we need a flexible solution that minimizes engineering costs or material costs.

The risk of adopting communications technology, which later proves unpopular and results in users demanding change to the alternate technology, is also a consideration. The NRE costs for switching a product to a standard will be weighed against the benefits of how early communications capabilities will be available.

After accounting for all the costs associated with networking an embedded device, we can move on to tackling issues such as the appropriateness of single transport, processor, networking protocol, or human or

machine interface. The solution lies in creating an economically viable solution that provides a single mechanism to interface with devices and accommodate the methods of transport—any device, running across any transport, to communicate with any application or interface.

2.0 The Solution: Internet-based Communications System to Interact with Devices

Although networking devices for the purpose of distributed control and data collection is not a recent phenomenon, the popularity of the Internet has driven massive development of interface and other networking software. The resulting tools and general awareness of networking has made widespread device networking much more practical than it has been in the past.

The benefits offered by networking embedded devices fall into the following four general categories:

Low cost user interface. A physical keypad and LCD display costs more as features and capabilities are added. In some cases, additional product features may make a device interface difficult to understand due to a proliferation of buttons and indicators. A browser-based interface, however, can be layered into simple control panels that relate to the current state of the device, and can be configured for an individual's needs.

Links to information available on the Internet. Product documentation and upgrade information may be integrated with the user interface. This can be much more helpful than a 'click here for the manual' system. The interface application can query the device and offer meaningful help and upgrade information. A historical database may be accessed to provide expert system features. An example would be a car diagnostic interface that would be greatly enhanced by information from other vehicles of the same type. Another benefit of integrating applications with available Internet resources is that new product features may be created. For example, a sprinkler controller enhanced with weather prediction and historical data will allow conservation based on weather predictions.

Value of information on device not previously available. Products that use embedded controllers often have some information that could be useful. As trivial as it may sound, things like the temperature of a furnace may indicate the need for filter replacement. Similarly, the duty cycle and outside temperature of a refrigeration unit may indicate a cleaning is needed. These are not exciting product features that drive consumers to replace old equipment, but they may reduce the cost of ownership and improve the product. The remote availability of performance and status information will help in diagnostic and configuration support. Information in the device may be useful in terms of providing usage information to marketing and advertising groups. A network of vending machines, for example, will provide feedback regarding the effectiveness of an advertising campaign. The same usage information may be needed to schedule restocking of product that has been vended. Finally, in systems whose purpose is the collection of data, remote connections reduce the price over manual collection efforts. The need of the power metering industry best illustrates this point.

Value of remote device control. The convenience of having remote access and monitoring of various systems can be an important product feature. In some situations, a product may not have an interface appropriate for interaction with a microcontroller, so the addition of remote access allows manipulation where none was previously possible. The embedded firmware engineer is sometimes faced with the responsibility to implement or choose a configuration that is the best setting for most users of a particular product. Once chosen, this setting becomes an immutable product characteristic. Access to configuration values provides a mechanism to provide product that more closely fits the user's needs. A device control panel may be viewed as such a decision. If an owner of a particular device could toggle between a simplified or sophisticated interface, the *world would be a better place*.

The points mentioned above illustrate an important point—the benefit derived from the communication capability may effect the parties involved with maintenance and support of a product, not the end user. This becomes important when assessing the value these new features have in influencing purchase decisions.

2.1 Increasing the value of Communication capabilities in a device

Networking technology has some benefits that are derived from the wide availability of other networked devices. Although these would not currently tip the cost/benefits scale, these factors will gain in impor-

tance as more devices become available through a network.

- Use a common client interface for devices from various vendors, or for various models of a device from a single vendor. When a particular interface for a device provides access, but does not expose features or controls that are unique to the device, then the development of a mechanism for using a common client interface will save product development costs. This seemingly obvious point, which has driven development of device driver architectures, database object models, and TCP/IP application services, seems to have had very little impact on device control architectures.

- The machine interface of a device, in the form of a communications API, allows the device to be incorporated into a larger control system which may be composed of many other embedded devices. The ability for a particular device to participate in such a system may be a product feature that increases the perceived value of the product in the eyes of an end user. The behavior of multiple systems may be coordinated in new ways. For example, having a car stereo fade to a lower volume when a cellular phone call comes or load shedding during peak power demand times.

- The value of communications to an embedded device can be real even with only one device. A Weiser Lock (see figure 1) can be of value without any other devices connected to the same subnetwork (i.e. a remote cabin can be programmed with temporary access for weekends renters.) There should not be a high cost to implement a small network (or a single device network) as a larger network. At the same time, if the Weiser Lock is added to a security system device on the same network, the security system can now show you that the door is closed and locked.

After evaluating the cost of implementation and value of access, it may be determined that no suitable connection technology exists for a particular application. It is our task to evaluate the world of embedded applications in terms of cost/benefits and create a strategy for technology development that will facilitate adoption of embedded networking technology.

The value of the communication function will dictate the set of available communications hardware and software solutions. It is unreasonable to expect a solution that costs more than the perceived value of the connection to be adopted.

3.0 The Gateway Approach

A gateway system provides an embedded architecture that addresses the issues discussed above and provides networking capability to even the smallest of embedded controllers. By establishing a gateway that acts as an intermediary "broker" between lightweight device networks (RS232, RS485, Modem, IR, RF) and heavyweight networks, including intranets and the Internet, you off-load the device and achieve the desired communication. The gateway can reside on a PC, single board computer, handheld, or a device with sufficient resources (32-bit microprocessor). The gateway, having more resources, can augment a device and enable more information and data to be exchanged.

A gateway provides device access and management services to nodes on the Internet or an intranet. This gateway is a TCP server that has the following set of responsibilities:

Device Firewall. The devices rely on a gateway process to provide authentication and encryption where appropriate. In addition, the gateway can augment the security on the device by enforcing the most current security policies in this rapidly changing, Internet-connected world. Establishing security to protect the embedded device from unwanted users is necessary, but it is also critical to protect the network from undesired behavior of a device. If a device has unrestricted access to services on a network, then the device may pose a security risk. Whether through malice or oversight, a device could compromise the integrity of a secured subnet. The device gateway is charged with the task of securing access to a device and enforcing a usage policy on the embedded application.

Protocol Translation. The gateway is responsible for adapting to the variety of device connectivity hardware and software solutions and delivering a uniform TCP/IP based access service. Thus, devices with existing communications software and hardware may be made available to network-based clients.

Device Watchdog. A service of the gateway is monitoring device status. Whether the devices are to remain connected at all times and the gateway reports failure of the ancillary network or device, or the device connection is required to periodically contact the

gateway process, the gateway reports to a designated client when a device becomes unavailable.

Event Handling. A particular event from a device may require a launch of an application, or dispatch of a pager or e-mail message. In this situation, there is no connection to a "client" waiting for a message.

Device Services. Instead of viewing the network as a client of an embedded application, it may be desirable to offer a set of network services to an embedded application. The gateway acts as a provider of services to embedded application. These services include database storage and retrieval, communications to network applications and other devices, device application extensions, and firmware updates. Some applications, such as automobiles and remote monitoring equipment, do not lend themselves to permanent hardwired connections. These mobile and remote applications may have persistent network storage in the form of a database, which may store commands, state, and history information that is always available to client applications. By de-coupling the device from a client that is interrogating the device state with a state database, the reduction in redundant traffic to the device is reduced and a point of interface for clients wishing to connect to a device may be down. A stock interface may respond and deliver information regarding the unavailability of the device. The batch- oriented paradigm, where the user essentially says "change the following settings next time it checks in" is coupled with a watchdog function and results in more efficient use of machine and human client's time. While this may be implemented in a standard email system, some extensions would be desirable, including the ability to signal the device, when it calls in, to remain connected while a client, that has previously requested a direct connection, is alerted.

Five aspects of a gateway-based system to consider include the following points.

3.1 Any Processor

There are more than 1,000 "flavors" of microcontrollers currently on the market. Each one is designed to cut costs and service a certain set of functions. There is no single processor or processor family that can service all embedded system requirements. In many cases, assumptions are being made in the embedded market that higher-end processors are required for any form of communications external to the embedded device – but the major logic flaw is that this is OK due to the ever-lowering cost in higher-end processors due to Moore's Law.

Moore's Law is being misapplied. We can't assume that 32- and 64-bit processors will drop so low in price—to pennies—and still deliver incredible performance. Even if cost is not an issue, there are still the issues of power consumption, package size, thermal dissipation, and other "embedded" requirements that are difficult, if not impossible, for larger processors to satisfy. At the same time, Moore's law also applies to all microcontrollers. These smaller MCUs are gaining additional capabilities while also costing less. Every device and application has different needs, therefore, a large array of processor choices with different performance, power requirements and capabilities is critical.

3.2 Any Capillary Network

Lightweight, capillary networks using transports such as RS232, dial-up, 485, RF, IR and CAN must all be supported to provide adequate options for networking 8- and 16-bit devices. Just like embedded processors, there is no single physical network transport that is appropriate for all environments. Whether it is cost, communications robustness, security, or a myriad of other reasons, solutions for networking embedded devices must support a wide variety of networking types that offer a range of capabilities. In addition, networks for 8- and 16-bit-based devices need to be able to operate in low-power, transient environments where devices may or may not have continuous access to a network. The method by which these lightweight networks communicate with larger, wide-area networks, including the Internet, must be efficient and timely. Unlike Ethernet and TCP/IP, which are proving to be an effective single-communications method for larger computer systems, there is no single lightweight network that is appropriate for smaller electronic devices.

3.3 Flexibility

Flexibility - EMIT software architecture is flexible and scalable. EMIT software can be configured for a wide variety of applications, including existing devices and proprietary networks. The user interface can operate on a remote Web browser, a directly connected laptop, or even a handheld PDA. Where a user interface is not necessary, it can connect directly to an application or database. By utilizing a gateway approach, implementers can also have flexibility on where inter-

face descriptions are stored. With a gateway, the interface can be fully described and stored on the gateway, eliminating any resource requirements for storing the interface on the device. Or, the gateway can be used to augment partial interface description such as a tagging method. emWare's® patent-pending Microtags are stored on the device yet reference pre-built interface components—such as emWare's emObjects—that can be called from the device but stored on the gateway.

3.4 Any Interface

There are countless methods to interface with devices including Windows, the Web browser and the phone. There is no such thing as the "ubiquitous interface" for smaller electronic devices. For certain types of applications, a simple, text-based HTML interface may be sufficient, however, in other types of applications it may be overkill or not sophisticated enough. To provide rich, device-networking solutions, human and machine interface options need to be provided that scale from DTMF—phone key—to complex enterprise databases and resource planning applications—such as SAP R-3. In addition, many of these "interfaces" may be used with these devices at the same time.

For example, emWare recently demonstrated a series of networked vending machines at an SAP worldwide conference. This demonstration, built on emWare's architecture, was a collaborative effort by SAP, IBM, Sybase, 3Com, and Micron. The vending machines could be accessed by a Web browser. In addition, the vending machine interfaced concurrently with a number of other interfaces and applications. IBM demonstrated voice recognition technology being used to "drop a can" (from the vending machine) through the emWare connection. Likewise, Sybase showed database integration for inventory management, and SAP showed enterprise application integration for scheduling service routes to the vending machines. 3Com's Palm Pilot was used through an IR interface to directly connect with the vending machine and get and set configuration parameters. In addition, Micron's near-field ID badges were used to demonstrate contactless, microcash transaction with the vending machines. They all leveraged emWare to allow their unique interfaces to interact with the vending machine device at the same time.

3.5 The Object Model

An object model for lightweight devices is required to accommodate the diversity of embedded devices, the different transport systems and collaboration requirements, creating the building blocks for making the integration of multiple object environments (CORBA, JINI, and DCOM) possible. This will allow for even greater sharing of information and control.

The end result for embedded device communication should be an object representing the capabilities of the device. Everything that the application engineer wants to make public is exported from the embedded application to a device object server and then through a Gateway to applications on the Internet or an intranet.

In many applications it is desirable to treat embedded devices as abstract data types. By encapsulating functional behaviors of a particular device type, and standardizing around these encapsulations, we create an environment that will foster the development of network-client applications. These applications use an object model and definition of a particular application function to de-couple the client interface from the manufacturer's device. The gateway provides access to the device through an object service, and isolates the client from network and device-specific considerations. A home management software package, which could run on a PDA or browser, may access a heating control unit based on the object definition of an HVAC. The gateway provides the necessary object interface to an HVAC and isolates the client from the specific network and device characteristics.

Several choices exist for representing embedded devices. These choices include open standards like CORBA, and proprietary standards including Sun's Jini, Microsoft's DCOM, and one currently in development at emWare that is targeted for smaller embedded devices. All of these systems require that industry-standard object definitions be created in order to abstract the functionality of a particular device. Like all of emWare's architecture, the object-oriented implementation does not impose large resource requirements on the device. However, emWare's object definition can also be linked with Jini, CORBA, or DCOM to provide enterprise-wide, object applications.

4.0 emWare's EMIT®

For example, emWare has developed Embedded Micro Interface Technology (EMIT) that provides developers and manufacturers with a cost-effective and flexible means to easily implement Internet capabilities into their 8- and 16-bit microcontroller-based devices. The three main components—emMicro, emGateway, and the interface application—provide a complete and flexible device-to-user solution.

emMicro is an extremely thin, device object server that requires minimal resources at the device by taking up as little as 1 Kb of space at the device. The exception-handling mechanisms for the object provides a way to signal alarms, request services, and perform other device-initiated actions. With the addition of emMicro in an embedded device, the device is now instructed to expose any desired variable, event, function, or document from the device to any application or interface via a network object methodology. emWare sees distributed network objects as a profound refinement in communications technology for embedded devices.

emGateway compliments emMicro by serving as a bridge between the lightweight device network and the Internet or intranet. emGateway also provides the additional services for multiple-device management and Internet communication with a standard Web browser, or other interface applications or databases. The Web browser can use pre-built interface components (emObjects) to display and interact with the device in an intuitive, straightforward manner without requiring large amounts of resources on the device. emGateway can also be distributed over the network, allowing for many communications and resource-shedding options. Other situations will require that emGateway be combined with a Web browser (on a laptop, for example) to directly connect to the device. The flexibility of EMIT's architecture allows the components to be distributed and installed where they make the most sense, depending on the situation.

emWare's goal is to identify the parts of the embedded application to export, then create a network object that represents this interface to the device. Given that, we must question the utility of implementing this on top of a large communications stack that is made general for the purpose of simultaneously supporting other communication paradigms. More traditional and general implementations result in higher costs and device requirements.

5.0 The Next Steps

The next frontier is to move beyond simply communicating with devices and create an environment that enables devices to be treated interchangeably. For example, this step requires an extensible architecture that enables an interface to be created for any device, such as an HVAC thermostat, without the environment needing to know the chip, manufacturer or vendor of the device.

5.1 A Collaborative Effort

Delivering on the promise of device network technology and all that it has to offer the world requires a tremendous collaborative effort. The cooperative efforts of microcontroller, software, hardware, database, enterprise systems, and user interface companies is required to bring all the pieces together in a cohesive working solution. For this reason, emWare has formed the "Embed The Internet" (ETI) Alliance to provide a forum for collaborative efforts between some of the largest companies in the world, including Hitachi, Philips, Analog Devices, Microchip, Atmel, Siemens, Mitsubishi, National Semiconductor, Motorola, SAP, Sybase, Phytec, Centura, TASKING and Pervasive. emWare has also started a working group to explore standards opportunities in the 8 and 16-bit microcontroller-networking market.

6.0 Summary

Bringing the Internet to all electronic devices, especially the billions of 8- and 16-bit devices, is a seemingly daunting challenge. The size of devices, the kind of processors, the type of transport and the information or data to be extracted from devices varies greatly. Solutions to date have been too specific and addressed just one of the variables, such as language, wire, transport or OS. The gateway system with its modular architecture, distributed approach and complete integration with microcontrollers, software, hardware, databases, user interfaces, and enterprise systems provides the framework for virtually any device networking implementation. This is accomplished because the gateway system enables the capabilities of an embedded device to be exported in the form of a network object. This object can then be integrated into network-based applications. Finally, the future of networked embedded devices lies in a collaborative effort to enable any device, in any industry to communicate over any network.

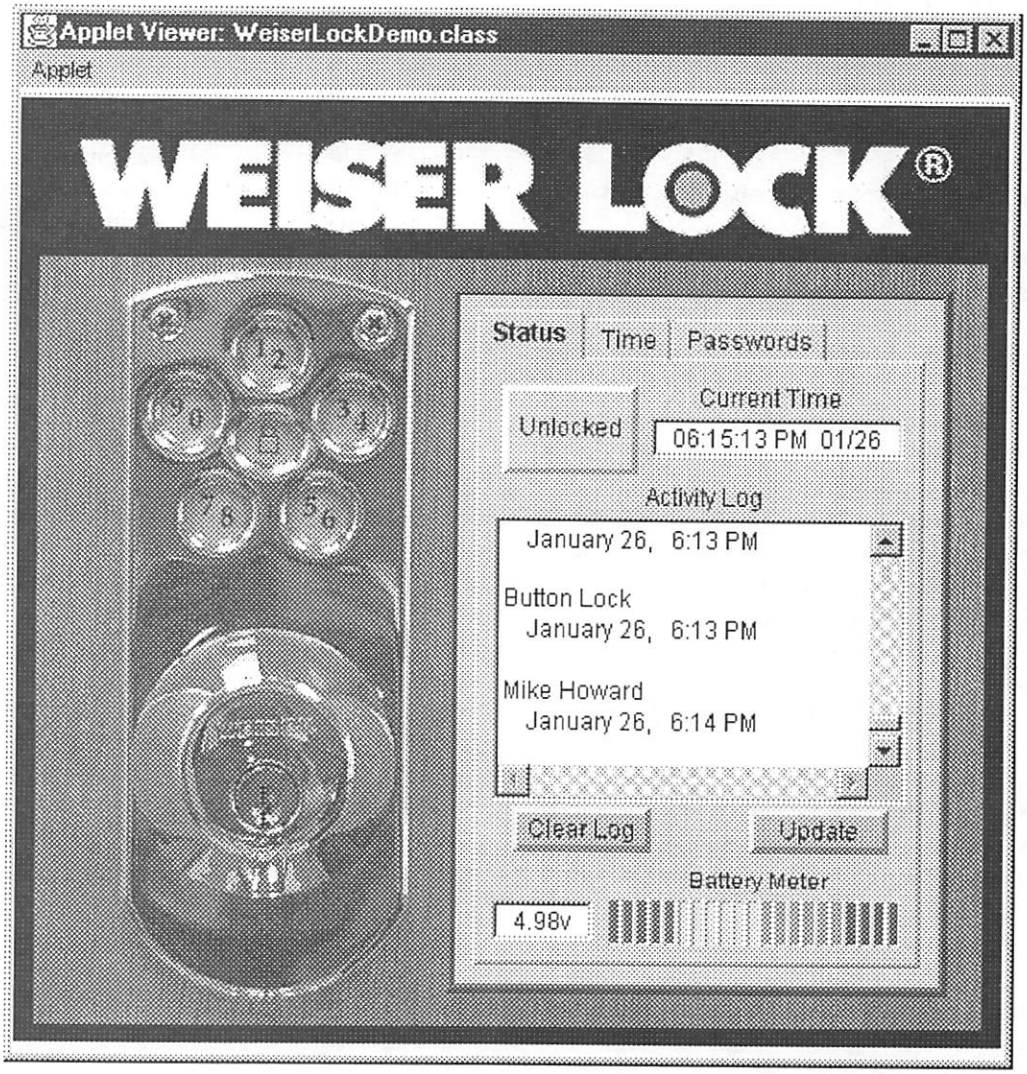

Figure 1 – Weiser Lock Interface

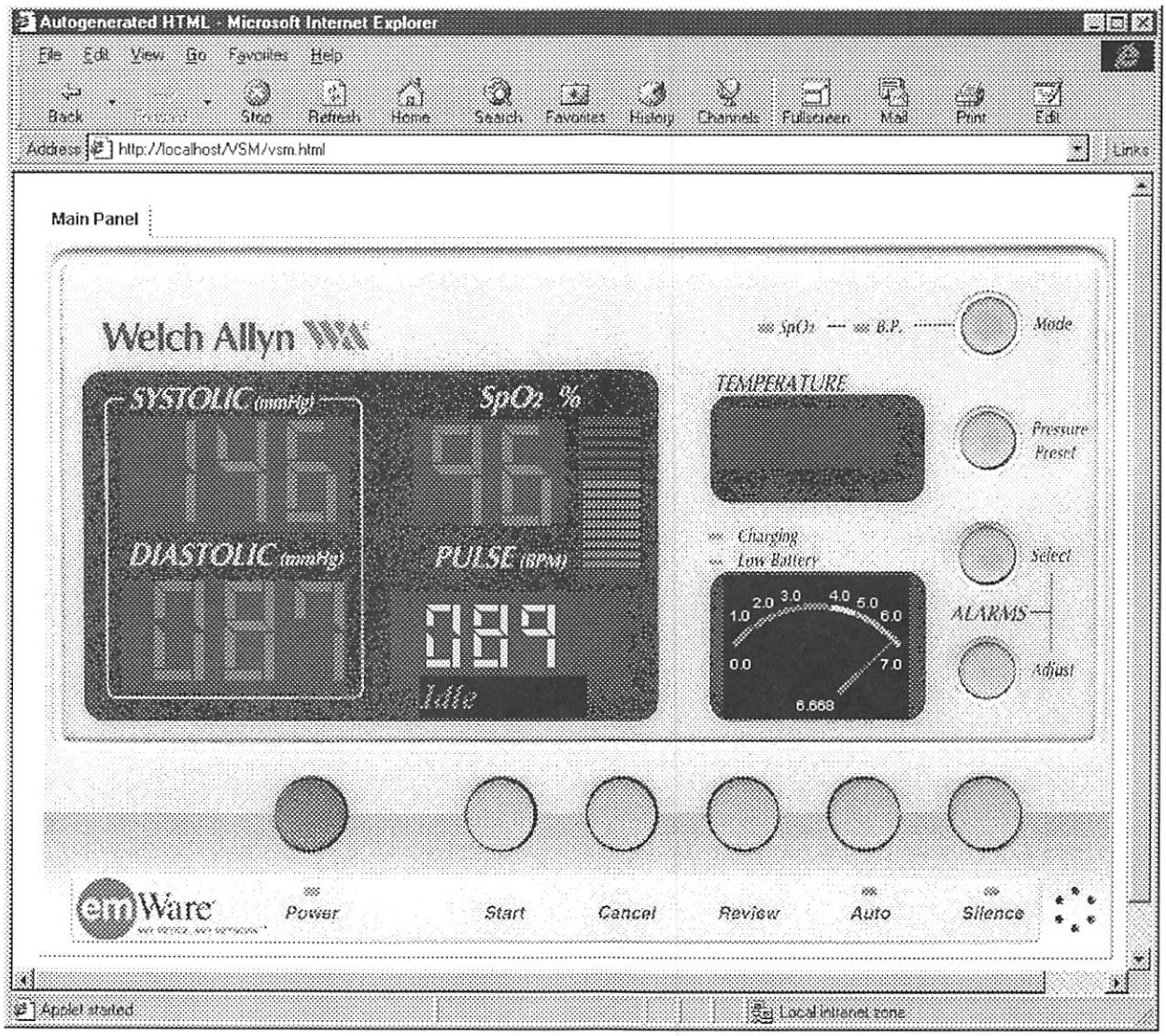

Figure 2 – Welch Allyn Medical Device

RETHER: A Software-Only Real-Time Ethernet for PLC Networks

Tzi-cker Chiueh
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

chiueh@cs.sunysb.edu http://www.ecsl.cs.sunysb.edu/~chiueh

Abstract

The networking technologies used in industrial automation are required to support real-time performance guarantees to ensure that sensor/command data reach target nodes within a delay bound. With the continuing popularity and thus the accompanied price drop of the Ethernet technology, replacing the typically closed and proprietary automation networks with Ethernet is emerging as a very attractive solution, because of the cost-effectiveness and its compatibility with the trend of moving industrial control systems to PC-based hardware. Due to its contention-based media access control protocol, Ethernet theoretically can not bound the network access delay without additional traffic control mechanisms. In this work, we present the design, implementation, and programming interface of a software-based traffic control protocol called *RETHER* , which turns off-the-shelf Ethernets into real-time networks without any hardware modifications. *RETHER* works both in a single-segment as well as a multi-segment environment. This paper reports the performance measurements and implementation experiences with the *RETHER* prototype, which has been fully operational for over 12 months.

1 Introduction

The basic building blocks in industrial automation systems are programmable logic controllers (PLC), dedicated control devices that interface with the physical instruments or sensors. Traditionally PLCs are connected together and with central control computers through a closed and proprietary network technology, because of special timing and/or hardware constraints. As Ethernet dominates the desktop computer world, its price is dropping precipitously. For example, 10 Mbits/sec ISA-based Ethernet adapters cost about $15, and yet 10 Mbits/sec is abundant for automation control applications. Moreover, as PLCs are themselves evolving towards PC-based hardware, choosing Ethernet as the backplane network technology makes even more sense.

The only problem is that commodity Ethernet hardware can not provide any performance guarantee, for the following two reasons. First, Ethernet's media access control protocol is CSMA/CD, which relies on an exponential backoff algorithm to resolve link collision among multiple nodes when they attempt to send data simultaneously. Due to the probabilistic nature of the exponential backoff algorithm, network access delay is inherently nondeterministic. Second, Ethernet does not support prioritization of packets. This means that time-critical network packets could be held up waiting for time-insensitive packets.

The *RETHER* project set out to develop an efficient delay/bandwidth bandwidth guarantee mechanism over off-the-shelf Ethernet without any hardware modification. With *RETHER* , industrial automation systems could use commodity Ethernet as the underlying network, thus reaping the benefits of economies of scale from the PC industry. *RETHER* is designed to be a software-only solution that is built into the device driver of the host operating system. Because it is part of the device driver, *RETHER* is transparent to higher-level network protocols such as TCP/UDP and IP. Consequently, all existing network applications can con-

tinue to run on a *RETHER* network without any changes. New real-time applications have to be specifically written against the API provided by *RETHER* , which is based on the industry-standard socket interface.

The design of *RETHER* also minimizes the performance overhead associated with supporting delay/bandwidth guarantees. In particular, *RETHER* features a hybrid mode of operation that automatically switches an Ethernet network between the *RETHER* mode and the CSMA/CD mode, depending on whether there are real-time connections active at the time. In the case of no active real-time connections, the network operates according to the CSMA/CD protocol, thus providing the same performance to non-real-time applications. To avoid starvation, *RETHER* ensures that a certain amount of bandwidth be reserved for non-real-time traffic. The reserved amount is chosen so that existing higher level network protocols such as TCP or NFS would not time out unnecessarily, and thus are to a large extent isolated from the existence of *RETHER*

The rest of this paper is organized as follows. In Section 2, we describe the programming interface and performance guarantee mechanisms of *RETHER* in the single-segment Ethernet environment. Then we use the single-segment *RETHER* protocol as the building block for multi-segment *RETHER* in Section 3. In Section 4, we present the performance measurements and implementation experiences of *RETHER* from our operational prototype. Section 5 reviews related work in this area to put the contribution of *RETHER* in perspective. Section 6 concludes this paper with a summary of the *RETHER* project and the current status of the report.

2 Single-Segment RETHER

2.1 Overview

RETHER provides a set of procedural interfaces for applications to reserve network bandwidth, and guarantees the reservations throughout the lifetime of the applications once they are admitted. Currently *RETHER* is implemented within the Ethernet device driver under FreeBSD UNIX, Linux and DOS. Under *RETHER* , individual applications can send a certain amount of data (D) in each periodic cycle, T. The amount of data is specified in the reservations and thus varies from application to application. The length of the cycle is fixed but can be changed at the system initialization time. To start a *RETHER* connection, which is *unidirectional*, the sender application makes a library call, `reservation()`, which, upon successfully creating a *RETHER* connection, returns a socket descriptor. From that time on, the sender application can `send()` the reserved amount of data through the returned socket descriptor, and *RETHER* ensures that the data be delivered at the rate $\frac{D}{T}$. The receiver application simply receives the sent data using normal `receive()` calls without making any special arrangements.

The current implementation of `reservation()` is based on the socket interface and uses a special unused Ethernet address[1] to alert the local *RETHER* module in the device driver of the reservation request. Upon detecting such a request, *RETHER* reserves a *RETHER* connection to the target receiver according to the requested bandwidth requirement, and if successful, records a mapping between the reserved *RETHER* connection's ID and the source port number associated with the socket descriptor to be returned to the sender application. At run time, *RETHER* consults with this mapping to determine which *RETHER* connection each IP packet should be sent on. Packets that are not parts of reserved connections are sent on a pre-established *RETHER* connection by default.

In the absence of real-time connections, the *RETHER* network operates under the original CSMA/CD protocol. When the first real-time bandwidth reservation request arrives, the *RETHER* network switches to a *token-passing* mode. In this mode, channel access for *all* traffic, real-time and non-real-time, is regulated by a token. Intuitively time is divided into cycles. In each cycle, the token first services all the nodes that have real-time data to send (called *real-time (RT)* nodes), and then it attempts to service *non-real-time (NRT)* nodes in a round-robin fashion. Essentially NRT nodes share the bandwidth remaining in each cycle after all RT nodes have been serviced. Note that *all* network nodes, including those making bandwidth reservations, are NRT nodes. The network stays in the token-passing mode until the

[1] `reservation()` itself uses a special IP address, which is mapped to the special Ethernet address through an artificial and persistent ARP cache entry.

last real-time session terminates. At this time, the network switches back to the CSMA/CD protocol. This hybrid scheme reduces the performance impact due to token passing on non-real-time traffic.

2.2 Token Passing

The token circulates in cycles so that real-time connections can access the network periodically. Because *RETHER* does not assume a globally synchronized clock, the token cycle time is maintained as a counter called the *residual cycle time* in the token itself. At the beginning of each cycle, the *residual cycle time* is set equal to a full token cycle time[2]. When the token visits a node, the node subtracts its token-holding time from this counter. Once the residual cycle time reaches zero[3], the token is passed back to the first real-time node and a new cycle begins. Figure 1 shows an example token visit schedule within a token cycle.

The token-holding time at each node is based on the amount of data the node is allowed to send. For real-time nodes, the bandwidth reservation determines the amount of data that needs to be sent out during every cycle. For non-real-time nodes, the amount of data that can be sent out is limited by the the total unreserved bandwidth and the size of messages in their output queues. *RETHER* has incorporated a mechanism to ensure that all the nodes use the unreserved bandwidth fairly. When the token visits a node, if it does not have data to send, it merely hands the token to its successor after subtracting the time to process the token.

2.3 Fault Tolerance

As the control token represents a single point of failure, *RETHER* incorporates a built-in fault tolerance mechanism to ensure continued network operation despite token loss due to machine failures, or token corruption due to random bit errors. Each *RETHER* node is required to monitor the health of its successor in the token passing schedule. When a node N sends the token to its successor S, it starts an acknowledgment timer waiting to hear from S. If S is alive, it sends an acknowledgment back to N when it sends the token forward to its own successor. If the successor node is dead for some reasons, the timer at the monitoring node times out and it pings the successor to ensure that the successor indeed dies. This extra ping is necessary to check if the successor is still alive but actually drops the token due to reasons like bit errors. On detecting a failure, the monitoring node broadcasts a message announcing the failure and regenerates a new token. The choice of the timeout value and other failure scenarios are discussed in greater detail in [10]. Although token recovery can take place within the same token cycle, in practice, it takes a little bit longer than one token cycle to recover. This is dependent on the precision with which we can set the acknowledgment timer value in the operating system[4]. *RETHER* also addresses many other failure scenarios such as multiple node failures in [10]. When a new node boots up, it broadcasts a message identifying itself. The node currently holding the token adds the new node to the list of live nodes maintained in the token. As a result the token will visit the new node in the next cycle.

2.4 Admission Control

RETHER incorporates a distributed admission control scheme that is guaranteed to be free of race conditions. The admission decision at a node is postponed until it receives the token, because the token carries the most up-to-date information about all active real-time and their bandwidth reservations. Since only one node holds the token at a time, mutual exclusion is automatic. A disadvantage with this scheme is that bandwidth reservation requests are delayed due to the waiting for the token to arrive before the admission decision can be made.

RETHER intends to support both real-time and non-real-time traffic on the same Ethernet segment. To prevent starvation for non-real-time traffic, only a fixed fraction of the total raw bandwidth is set aside for real-time connections. The reserved bandwidth for non-real-time traffic minimizes unnecessary timeouts for existing network protocols such as NFS, and places an upper bound on the token inter-

[2]This is a system initialization parameter and can be, for instance, set to 33 msec for applications requiring 30 video frames per second.

[3]In practice, the token is passed back to the first RT node when a non-real-time node determines that the residual cycle time in the counter is less than that needed to send out the packet at the head of its message queue. Therefore, the token cycle counter need not be zero when a new token cycle is started.

[4]Most UNIX systems' timer resolution is 10 msec.

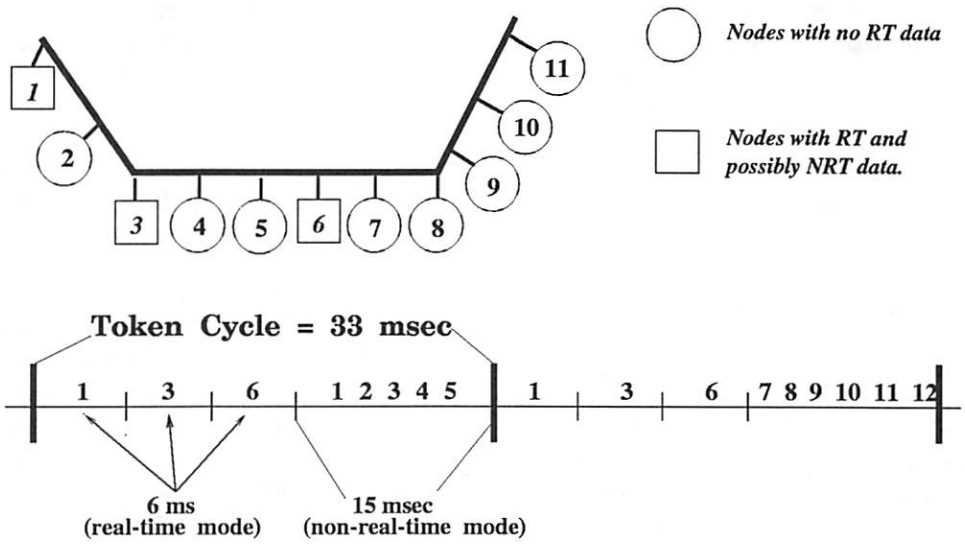

Figure 1: *Node 1, 3, and 6 are real-time nodes, each of which is assumed to have a token holding time of 6 msec every time the token visits them in the real-time mode. As every network node has non-real-time data to send, ALL network nodes are non-real-time nodes. In this example, the token visits real-time nodes in the first 18 msec of the 33-msec token cycle, and then visits non-real-time nodes in the rest of the cycle in a round robin fashion. Note that the token continues the visiting schedule for non-real-time nodes in the next cycle (Node 6) from where it left off in the previous cycle (Node 5) .*

arrival time as seen by NRT nodes. This bound is critical to a timer-based mechanism for detecting and recovering from disastrous failure scenarios.

3 Multi-Segment RETHER

Due to electrical signal considerations, multiple Ethernet segments connected by bridges or switches are required to accommodate a large number of PLC nodes. To provide end-to-end network bandwidth guarantees between any pair of nodes in a multi-segment Ethernet environment, *RETHER* has been extended to operate across switches. Conceptually, a real-time connection applies the single-segment *RETHER* protocol to reserve on each of the segments on the path from its source to the destination. The per-segment reservations along the way are parts of one logical real-time connection. Consider the network configuration in Figure 2. Suppose a real-time connection runs from A1 to C1. This connection is broken down into three sub-connections namely A1-Gw1 on Segment 1, Gw1-Gw2 on Segment 2, and Gw2-C1 on Segment 3, where the nodes on the left are the sender and those on the right are the receivers. For a multi-segment real-time connection, each intermediate switch acts as a sender on one segment and a receiver on the other. All Ethernet segments run the single-segment *RETHER* protocol, with the token cycles on different segments proceeding completely independently of one another. That is, token cycles associated with adjoining segments, although of the same length, are not synchronized at all.

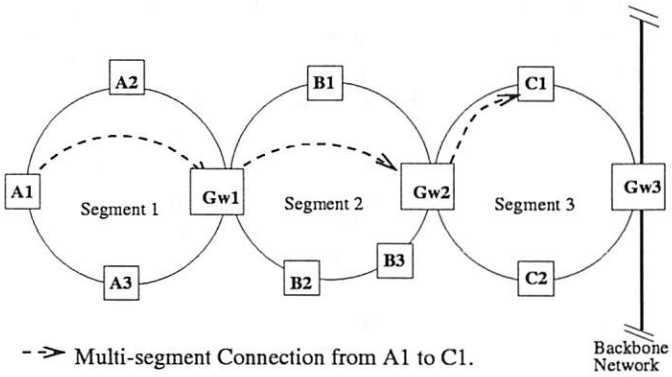

Figure 2: *A sample multi-segment* RETHER *connection from Node A1 to C1, through two intermediate switches Gw1 and Gw2.*

3.1 Connection Setup and Admission Control

In single-segment *RETHER* , connections are set up by modifying the information on the token when it arrives. It is not necessary to contact any other node, including the receiver. Multi-segment *RETHER* , however, needs an explicit connection setup protocol to establish end-to-end real-time connections across bridges/switches. *RETHER* connection messages are initiated by the senders and routed via the static routing tables in the switches. As a multi-segment *RETHER* connection is established, resources, namely buffers and network bandwidth, on the nodes along the way are reserved accordingly. More concretely, as each intermediate switch receives a connection request, it creates a single-segment *RETHER* connection on the network link through which the message is forwarded. In addition, the switch maintains the following information:

- The multi-segment *RETHER* connection ID, in the form of the sender's IP address and source port number, and the local single-segment *RETHER* connection ID.
- The next-hop interface and the Ethernet address of the next-hop node.
- The previous-hop interface and the Ethernet address of the previous-hop node.

When the last-hop switch successfully reserves a single-segment *RETHER* connection to the destination node, it sends back an acknowledgment through the same path to *commit* the resource reservations made by intermediate switches and the sender. Only when the sender receives a success acknowledgment for connection establishment will it start to send real-time data out.

In multi-segment *RETHER* , admission control is performed independently on each of the segments on a hop-by-hop basis. Each intermediate switch applies the same admission control criterion as single-segment *RETHER* . If the admission test fails in any of the segments, a connection termination message is sent on the same path back to the sender node, to cancel all resource reservations. Note that each single-segment *RETHER* connection is only aware of its other end-point in the same segment. The information about the final source and destination of the multi-segment *RETHER* connection is hidden from non-periphery switches.

3.2 Fault Tolerance

When the token on an Ethernet segment is lost or corrupted, the single-segment *RETHER* protocol's fault tolerance mechanism recovers from the fault by reintroducing the token in that segment. All the real-time connections that pass through the segment continue to work after token recovery. Therefore, multi-segment *RETHER* does not introduce any new problems compared to single-segment *RETHER* in this case. However, when network nodes crash, new mechanisms need to be devised to handle multi-segment connections in which the failed nodes participate.

For a multi-segment *RETHER* connection, either the crashed node is involved in the real-time connection or it is not. If the failed node is one of the intermediate switch or an end-point of a *RETHER* connection, the state associated with the connection needs to be cleaned up and the connection has to be reestablished, if possible. Connection re-establishment only makes sense when the failed node is one of the intermediate switches and there is an alternative route that can be used to bypass the failed switch. The cleanup of the state associated with a *RETHER* connection whose intermediate switch has crashed is triggered by the detection of this failure in all the segments to which the switch is connected. Because a *RETHER* switch participates in the token passing on all segments that are connected to it, the switch failure is detected independently on each of the segments via the fault tolerance scheme built into the single-segment *RETHER* . The nodes that detect the failure then broadcast a message to that effect on their respective segments. The other end points of the sub-connection to which the crashed node was connected, upon receiving such a message, frees up associated resources, and sends an abort message to the next sub-connection. The message eventually reaches the actual end-points of the connection in either direction and all the reserved resources for the connection are released. Just like the termination message due to failure of admission, a message travels along the path of the connection on both sides of the failed node. If the failed node is an end point of a real-time connection, the processing is similar except the clean-up message for each af-

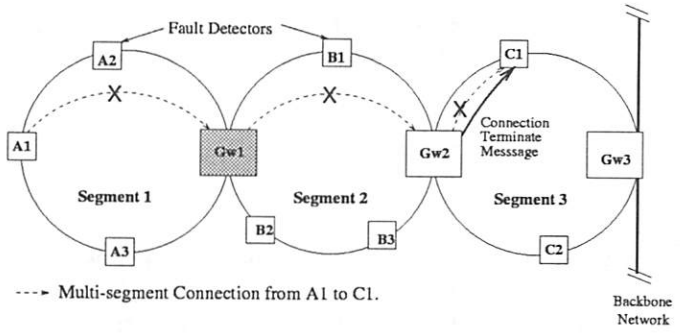

Figure 3: *Failure of an intermediate switch in a multi-segment connection. The failure is detected independently on all the segments the switch is connected to, in this case, Segment 1 and 2.*

fected real-time connection only propagates in one direction.

For instance, in Figure 3, suppose Gw1 were to crash and Node A2 detects the failure on Segment 1 and Node B2 detects it on Segment 2. A2 and B2 inform all the node on their respective segments by broadcasting a message. On receiving the message, A1 terminates its sub-connection and frees up the connection's associated resources. Gw2 similarly terminates the sub-connection whose other end-point was Gw1, on Segment 2 Since this is a multi-segment connection, Gw2 also sends a message to C1 to terminate the entire connection and to free the reserved resources. Thus, all the real-time connections that have Gw1 on their path are terminated with the associated state across the network cleaned up.

If, on the other hand, the failed node is not involved in the real-time connection, then the connection continues as before. For example, if Node A3 in Figure 3 dies, the real-time connection from A1 to C1 remains operative after the token recovery. The failure is detected and the token is recovered locally in the segment to which the failed node is connected. The effect on any real-time sessions crossing this segment is that they would not be able to send/receive data during the fault recovery period.

3.3 Buffer Management

The token cycles on adjoining network segments are not synchronized and this could lead to longer latency because of temporal skews of the token arrival times on connected segments. Since the token rotation times (TRT) in both segments are the same, the maximum skew between them could be one TRT long. For example, in the worst case, the switch receives data from the incoming segment but just missed the token on the outgoing segment. Hence, it may have to wait as much as one TRT before forwarding the data onto the outgoing segment. In the meantime, data would start arriving on the incoming segment for the next cycle, leading to buffer overflow if there is only one buffer at the switch. To avoid this situation, we use a double buffering scheme in which there are two buffers for each real-time stream going across the switch. While one buffer is being filled by the input sub-connection on one segment, the other is emptied out by the output sub-connection when the token on the outgoing segment arrives. At the end of each token cycle, the roles of the two buffers switch.

The size of the buffer has a direct effect on the end-to-end latency experienced by the applications. This implies that the buffering delays at each hop must be small and that the number of hops that a real-time connection can cross, is bounded. In theory, the token cycle times on all network segments are supposed to be the same, and in each cycle the switch receives exactly one frame of data from the incoming segment and sends out exactly one frame of data on the outgoing segment. Therefore, the real-time connection suffers a maximum of one token cycle latency at each hop along the path, and the worst-case end-to-end latency is *Number of hops * Token cycle time.* The minimum latency would be the time to transmit the data from the sender to the receiver as though they were on the same segment plus the time to copy the data into memory and out at each intermediate switch. Unfortunately, in practice, neither the network segments have identical token cycle times, nor does the switches forward the data they receives immediately.

4 Implementation and Performance

Both single-segment and multi-segment *RETHER* have been successfully implemented and tested on a prototype test-bed. The initial implementation was on FreeBSD 2.1, and has been ported to Linux and DOS. The test-bed, shown in Figure 4, consists of four Ethernet segments connected by three switches. Two of the segments are 100 Mbps Eth-

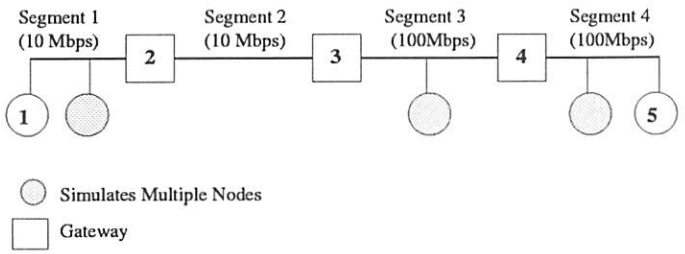

Figure 4: *The network setup for multi-segment* RETHER *experiments. There are four segments, two of them 100-Mbps and the other two 10-Mbps Ethernets.*

ernet while the other two are 10 Mbps Ethernet. The switch that connects the two 10-Mbps segments, and the non-switch machines on the 10-Mbps segment are 66MHz 486 machines, while those on the 100-Mbps segments are 90MHz and 100MHz Pentiums. Although the wiring and the NIC hardware at the hosts need not be changed, *RETHER* has to be implemented in the intermediate switches to provide bandwidth/delay guarantees. Because modifications to commercial Ethernet switches were not possible for us, we implemented the *RETHER* switch using a general-purpose machine that is equipped with multiple Ethernet interfaces, much like a network-layer software router. With the advent of faster microprocessors and system architecture, we believe implementing LAN switches based on general purpose machines is both feasible and cost-effective. Our implementation experience shows that it is indeed possible to build a *RETHER* switch completely in software. Since all the experiments are conducted locally in our lab, propagation delays are negligible in these measurements. For all the following measurements, the token cycle time is set to be 33 msec.

Extensive tests on the prototype demonstrate that bandwidth reservations made by *RETHER* connections are indeed satisfied in all cases. Since *RETHER* is implemented directly inside the device driver, each packet arrival, be it token or data, entails an interrupt processing overhead. When the network is lightly loaded and there are few nodes in the network, the token simply circulates around the network and the CPU processing overhead for token-circulation interrupts is significant. This is indicated in Figure 5. The graph plots the time taken by a user level process to execute the same computation intensive program without *RETHER* and with *RETHER* in the presence of minimal real-time bandwidth reservation. The measurements were

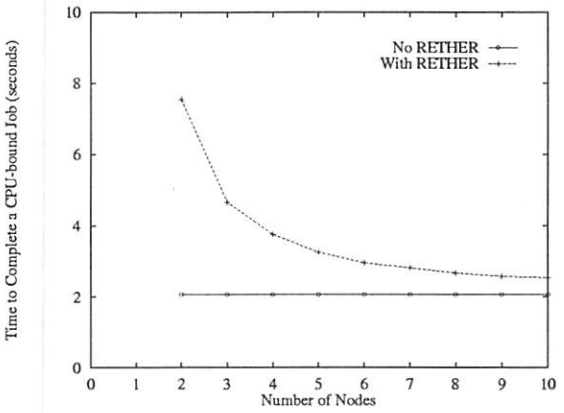

Figure 5: *As the number of nodes on the network increases, the per-node token-induced interrupt processing overhead decreases, because the token visits a node less frequently.*

made on a 100-Mbps network in which the token processing time is only 70μsec per node. As can be seen, the token-induced interrupt overhead becomes acceptable only when the 100-Mbps network has five or more nodes. On 10-Mbps networks, the relative interrupt processing overhead was not as bad because the token processing time is around 450 μsec.

No. of Hops	No Pre-existing Connection	With Pre-existing Connection
0	0.647	1.098
1	1.050	2.052
2	2.646	5.592
3	4.463	9.119

Table 1: *The time in msec to set up a real-time connection across different numbers of switches, with and without pre-existing real-time connections.*

Table 1 indicates the time to setup connections crossing 0 to 3 switches. Column 2 shows the connection setup time when all the Ethernet segments are in the CSMA mode. The main delay component in this case is the time to switch each segment from the CSMA to the *RETHER* mode. Column 3 indicates the time taken to setup a connection when the corresponding network segments are already running *RETHER* . In this case, the main component in the connection establishment time is the time to forward the connection establishment message in the non-real-time mode. The connection establishment time increases with the amount of bandwidth already reserved for real-time connections because it would take longer for the connection

request message, which is transmitted as non-real-time traffic, to reach its destination. The protocol processing associated with connection setup itself at each intermediate switch is relatively minor compared to the above times. A significant component of the connection setup delay is due to scheduling and executing user processes at either end-point to complete the connection establishment. However, these are not under the control of *RETHER* and thus are not included here. The times reported in Table 1 include the time to set up the connection at the receiver and sender ends in the kernel, but do not include any user-level processing.

5 Related Work

Kopetz's MARS system [4] prototype was also focused on process control applications, but used a TDMA protocol to provide real-time guarantees on Ethernet. described a multi-token-ring protocol that is designed The token ring in Totem [5] provided ordered multicasting rather than real-time performance guarantees. Hermant [2] presented a variant of CSMA/CDR for real-time scheduling in distributed multi-access broadcast communication channels. This protocol was not meant for existing Ethernet hardware.

More recently several commercial products available in the market attempt to provide real-time performance guarantee over LANs. HP's 100VG-AnyLAN [1] uses advanced Demand Priority Access to provide users with guaranteed bandwidth and low latency, and is now the IEEE 802.12 standard for 100-Mbps networking. National Semiconductor's Isochronous Ethernet [8] includes a 10-Mbps P channel for normal Ethernet traffic, 96 64-Kbps B channels for real-time traffic, one 64-Kbps D channel for signaling, and one 96-Kbps M channel for maintenance. The 96 B channels can provide bandwidth guarantee to network applications because they are completely isolated from the CSMA/CD traffic. Isochronous Ethernet forms the IEEE 802.9 standard. 3COM's Priority Access Control Enabled (PACE) [3] technology enhances multimedia (data, voice and video) applications by improving network bandwidth utilization, reducing latency, controlling jitter, and supporting multiple traffic priority levels. PACE technology uses star-wired switching configurations and enhancements to Ethernet that ensure efficient bandwidth utilization and bounded latency and jitter. Because the real-time priority mechanism is provided by the switch, there is no need to change the network hardware on the desktop machines. More recently, there are 802.1p and 802.1q efforts that support packet prioritization and virtual LANs. Peterson [6] summarized the current efforts in the Industrial Automation community to use Ethernet as the control network technology.

The difference between *RETHER* and all the above work is that *RETHER provides bandwidth/delay guarantees* to network packets, rather than just supports packet prioritization. In addition, all of the other schemes require changes to the existing infrastructure in the host network hardware and/or the wiring, while *RETHER* does not. *RETHER* 's ability to use commodity Ethernet hardware is crucial for industrial automation systems to ride with the technology momentum of the PC networking industry. Finally, *RETHER* is the only system that provides bandwidth guarantees for real-time connections that run cross multiple hops, an critical feature for system scalability.

6 Conclusion

This paper presents the design, implementation, and evaluation of single-segment and multi-segment *RETHER* protocols. The major contribution of this work is a simple and efficient traffic control mechanism that turns commodity off-the-shelf Ethernet hardware into a network capable of providing delay/bandwidth guarantees, and thus makes it possible to deploy Ethernet to PLC networks used in industrial automation systems. Because *RETHER* is a software-only solution, no hardware modification is required. *RETHER* is also innovative because it supports both real-time and non-real-time traffic in a single framework, and because it is scalable to a large number of PLC nodes with its built-in end-to-end performance guarantee mechanism.

RETHER has also been extended to wireless LAN with roaming support [7]. In addition, we have also developed a new real-time Ethernet switch architecture called *EtheReal* [9], which takes one step further by providing bandwidth/delay guarantees without requiring any changes to both software and hardware on the host ends. We are currently developing a middleware based on the distributed shared memory abstraction and *RETHER* , with the goal

to simplify the development of distributed real-time embedded systems. More up-to-date information about the *RETHER* project can be found in `http://www.ecsl.cs.sunysb.edu/rether.html`.

Acknowledgments

This research is supported by an NSF Career Award MIP-9502067, NSF MIP-9710622, NSF IRI-9711635, NSF EIA-9818342, NSF ANIR-9814934, a contract 95F138600000 from Community Management Staff's Massive Digital Data System Program, USENIX student research grants, as well as fundings from Sandia National Laboratory, Reuters Information Technology Inc., and Computer Associates/Cheyenne Inc.

References

[1] A.R. Albrecht and P.A. Thaler. Introduction to 100VG-AnyLAN and the IEEE 802.12 local area network standard. *Hewlett-Packard Journal*, 46(4), Aug. 1995.

[2] J.-F. Hermant, G. Le Lann, and N. Rivierre. A general approach to real-time message scheduling over distributed broadcast channels. *roceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, pages 191–204, Oct. 1995.

[3] The 3COM Technical Journal. 3Com's New PACE Technology. *http://www.3com.com/files/mktg/pubs/3tech/195pace.html*, 1996.

[4] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: the mars approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[5] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, C.A. Lingley-Papadopoulos, and T.P. Archambault. The totem system. *Digest of Papers, Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 61–66, Jun. 1995.

[6] C. Peterson. Open networking fuels the next major advancement in industrial automation. *EE Times*, Sep. 14 1998.

[7] P. Pradhan and T. Chiueh. Real-time performance guarantees over wired and wireless lans. *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, June 1998.

[8] Xiaonong Ran and W.R. Friedrich. Isochronous LAN based full-motion video and image server-client system with constant distortion adaptive DCT coding. *Proceedings of the SPIE - The International Society for Optical*, 2094:1030:41, 1993.

[9] S. Varadarajan and T. Chiueh. Ethereal: A host-transparent real-time fast ethernet switch. *Proceeding of International Conference on Network Protocols*, October 1998.

[10] Chitra Venkatramani. *The Design, Implementation and Evaluation of RETHER: A Real-Time Ethernet Protocol.* PhD thesis, State University of New York at Stony Brook, December, 1996.

Pebble: A Component-Based Operating System for Embedded Applications

John Bruno†, José Brustoloni, Eran Gabber, Avi Silberschatz, Christopher Small

Lucent Technologies—Bell Laboratories
Information Sciences Research Center
600 Mountain Ave.
Murray Hill, NJ 07974
{jbruno, jcb, eran, avi, chris}@research.bell-labs.com

†*Also affiliated with the University of California at Santa Barbara*

Abstract

The Pebble operating system is intended to support complex embedded applications. This is accomplished through two key features: (1) safe extensibility, so that the system can be constructed from untrusted components and reconfigured while running, and (2) low interrupt latency, which ensures that the system can react quickly to external events.

In this paper we discuss Pebble's architecture and the underlying technology used by Pebble, and include microbenchmark performance results on three MIPS target systems. The performance measurements demonstrate that Pebble is a good platform for complex embedded applications.

1 Introduction

This paper describes the Pebble operating system architecture, which is designed to be used as a platform for high-end, embedded, communicating devices constructed from reusable software components.

A *component-based approach* to building applications is becoming increasingly popular, as exemplified by the popularity of COM, Java Beans, and CORBA. The advantage of software components with clean, abstract interfaces is that they allow code to be combined and reused in ways that were not imagined by their developers, allowing the cost of development to be amortized over more uses.

When constructing a system from software components, a choice has to be made as to what method will be used to isolate the components from one another.

There are three basic approaches: provide no protection between components, as exemplified by COM-based systems; provide software protection, as used by Java-based systems; and provide hardware protection, as exemplified by CORBA- and RPC-based approaches. Each method has its drawbacks. With no protection, a misbehaved component can compromise the integrity of the system, as it has access to all data and resources of the system. Software protection typically requires that the system be written in a special, safe programming language, which may not be acceptable to all developers. Hardware protection schemes have traditionally exhibited poor performance, due to the cost of switching protection domains when performing an inter-protection-domain call. However, recent work (e.g., on L4 [Liedke97] and Exokernel [Kaashoek97]) has shown that commonly held assumptions about the intrinsic performance cost of hardware protection schemes need to be reexamined.

The Pebble architecture began with the idea that operating system services should be implemented by a collection of fine-grained, replaceable user-level components. The techniques we have applied in order to get good performance from operating system components are also used by component-based applications running on Pebble, and applications share in the performance benefit provided by these techniques. The performance improvement are striking: for example, on Pebble, a one-way inter-protection domain call takes roughly 120 machine cycles, which is within an order of magnitude of the cost of performing a function call; an equivalent call when running OpenBSD (on the same hardware) takes roughly 1000-2000 machine cycles [Gabber99].

With Pebble, an application can dynamically configure the services provided by the system, and safely load new, untrusted components, written in an unsafe pro-

gramming language, into the system while the system is running. Moreover, old servers may be retired gracefully when new versions of the service are introduced without disrupting the operation of the system. This capability is essential for high-availability systems that must operate continuously.

Future communication devices, such as PDA-cell phone hybrids, set-top-boxes, and routers will require just this type of dynamic configurability and the ability to safely run untrusted code. We feel that the approach that we are taking with Pebble will be valuable for building such embedded systems.

The remainder of the paper is organized as follows. Section 2 describes the Pebble philosophy. Section 3 contrasts software and hardware protection for component-based applications. Section 4 discusses the technologies used to implement Pebble, including protection domains, portals, scheduling, synchronization, device drivers and interrupt handling. Section 5 explains how portals are used to optimize file operations. Section 6 presents performance measurements of Pebble on three different MIPS-based target machines. The paper concludes with a survey of related work, current status and a summary.

2 Pebble Philosophy

The Pebble architectural philosophy consists of the following four key ideas. When combined, they enable our goals of providing low interrupt latency and low-cost inter-component communication.

- *The privileged-mode nucleus is as small as possible. If something can be run at user level, it is.*

The privileged-mode nucleus is only responsible for switching between protection domains, and other than synchronization code, is the only part of the system that must be run with interrupts disabled. By reducing the length of time that interrupts are disabled, we reduce the maximum interrupt latency seen.

In a perfect world, Pebble would include only one privileged-mode instruction, which would transfer control from one protection domain to the next. By minimizing the work done in privileged mode, we reduce both the amount of privileged code and the time needed to perform its essential service.

- *Each component is implemented by a separate protection domain. The cost of transferring control from one protection domain to another should be small enough that there is no performance-related reason to co-locate components.*

Microkernel systems, such as Mach, have in the past tended towards coarse-grained user level servers, in part because the cost of transferring between protection domains was high. By keeping this cost low, we enable the factoring of the operating system, and application, into smaller components with small performance penalty.

- *The operating system is built from fine-grained replaceable components, isolated through the use of hardware memory protection.*

Most of the functionality of the operating system is implemented by trusted user-level components. The components can be replaced, augmented, or layered.

A noteworthy example is the scheduler: Pebble does not handle scheduling decisions at all. The user-replacable scheduler is responsible for all scheduling and synchronization operations.

- *Transferring control between protection domains is done by a generalization of hardware interrupt handling, termed portal traversal. Portal code is generated dynamically and performs portal-specific actions.*

Hardware interrupts, inter-protection domain calls, and the Pebble equivalent of system calls are all handled by the *portal* mechanism. Pebble generates specialized code for each portal to improve run-time efficiency. The portal mechanism provides two important features: abstract communication facilities, which allow components to be isolated from their configuration, and per-connection code specialization, which enables the application of many otherwise unavailable optimizations. Portals are discussed in depth in Section 4.

3 Software vs. Hardware Protection

Suggesting the use of a hardware protection scheme, in the age of Java and software protection, is a controversial (perhaps heretical) proposal. However, each approach has its applications.

Component-based systems that use software protection schemes are typically written in a type-safe byte-coded language, such as Java [Gosling96] and the Limbo language of the Inferno system [Dorward97]. Components

run in a single hardware protection domain, but the run-time environment implements (in effect) a software protection domain.

These systems are typically designed with the following three goals:

1. Provide an architecture-independent distribution format for code.
2. Ensure that resources (such as memory) are returned to the system when no longer needed.
3. Ensure that the component does not view or modify data to which it has not been granted access.

In the case of Java, these goals are satisfied by (1) the machine-independent Java byte-code, (2) the garbage collector provided by Java run-time environments, and (3) the run-time Java byte-code verifier.

Java byte-code offers a hardware-architecture-neutral distribution format for software components. However, such an architecture-neutral format could also be used for untrusted code. Most compiler front ends generate a machine-independent intermediate form, which is then compiled by a machine-specific back end. Such an intermediate form could be used as a distribution format for components written in any language, trusted or untrusted.

Software protection has problems too. Putting all software-protected components in the same address space makes it hard to pin down a buggy component that is not caught by the type system or the garbage collector.

Hardware protection schemes run each component in a separate hardware protection domain. As an example, a traditional operating system (such as Unix) could be thought of as a hardware-protected component-based system, where the components are programs, and protection is provided by the operating system working in concert with the hardware memory management unit.[1]

Typically, hardware schemes do not attempt to satisfy (1), since components are distributed in the machine language of the target hardware. (2) is satisfied by careful bookkeeping: the system keeps track of the resources assigned to each component (process), and when the component (process) terminates, the resources are returned to the system. (3) is implemented using hardware memory protection: each component is run in a separate address space. If a component attempts to view or modify data outside its address space, a hardware trap is taken and the component (process) is terminated.

By running multiple Java virtual machines on top of hardware protection one can separate the components in a way that makes it easier to identify buggy components.

Software schemes are the only option when hardware protection is unavailable, such as low-end processors without memory management units. (The designers of Inferno/Limbo chose a software protection scheme for precisely that reason.)

Hardware schemes are called for when component code is written in an unsafe language, such as C or C++. Although Java provides many facilities unavailable in C and C++, there are often good reasons for running code written in an unsafe language. For example, a component may include legacy code that would be difficult or costly to reimplement in a safe language, or may include hand-tuned assembler code that uses hardware specific features. One example of the latter is an implementation of a computation-intensive algorithm (such as an MPEG decoder) using special hardware instructions designed to support such tasks.

The garbage collection facility offered by systems such as Java can be seen as both a blessing and a curse. On one hand, garbage collection frees programmers (and software testers) from ensuring that all allocated resources are eventually freed. Storage leaks are notoriously hard to find, and automated garbage collection greatly simplifies the task of writing robust code. However, when building applications that have fixed latency requirements, garbage collectors can be more trouble than they are worth. Because the free memory pool may become empty at any time, any memory allocation could trigger a collection, which makes estimating the cost (in time) of a memory allocation a stochastic, rather than deterministic, process. In a real-time embedded system, the uncertainty introduced by the presence of a garbage collector may not be worth the benefit it offers. This is the reason why some systems, such as Inferno, invested much effort to ensure that garbage collection does not delay critical functions.

One final historical argument for choosing software protection over hardware protection is that the cost of transferring between components under a hardware protection scheme is several orders of magnitude higher than it is under a software protection scheme. In this paper we show that the techniques used by Pebble bring the cost of cross-component communication under a hardware scheme to within an order of magnitude of the

1. To continue the analogy, such components can be composed using Unix pipes.

cost of a function call, which we feel makes this point moot.

4 Pebble Technology

In this section we discuss the technology used to implement Pebble.

4.1 Protection Domains and Portals

Each component runs in its own *protection domain (PD)* which consists of a set of memory pages, represented by a page table, a *portal table*, which holds the addresses of the *portal code* for the portals to which the component has access. A protection domain may share both memory pages and portals with other protection domains, as discussed below.

A parent protection domain may share its portal table with its child. In this case, any changes to the portal table will be reflected in both parent and child. Alternately, a parent protection domain may create a child domain with a copy of the parent's portal table at the time when the child was created. Note that both copying and sharing portal tables are efficient, since portal tables contains pointers to the actual portal code. No copying of portal code is needed in either case.

A thread belonging to protection domain A can invoke a service of protection domain B only if A has successfully opened a portal to B.[1] Protection domain B, which exports the service, controls which protection domains may open portals to B, and hence which components can invoke B's service. Protection domain B may delegate the execution of its access-control policy to a third party, such as a directory server or a name-space server.

To transfer control to B, A's thread executes a trap instruction, which transfers control to the nucleus. The nucleus determines which portal A wishes to invoke, looks up the address of the associated portal code, and transfers control to the portal code. The portal code is responsible for saving registers, copying arguments, changing stacks, and mapping pages shared between the domains. The portal code then transfers control to component B. Figure 1 shows an example of portal transfer.

When a thread passes through a portal, no scheduling decision is made; the thread continues to run, with the same priority, in the invoked protection domain.

1. "Protection domain A" here is shorthand for "the protection domain in which component A is running."

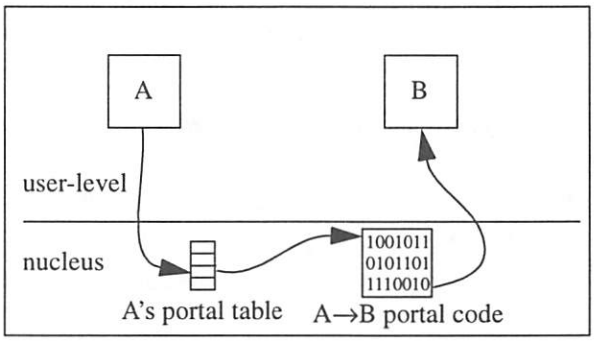

Figure 1. Portal Transfer. Protection domain A invokes protection domain B via a portal transfer. Protection domain A transfers indirectly through its portal table to the portal code specific to this communication path. The portal code transfers control to protection domain B.

As part of a portal traversal, the portal code can manipulate the page tables of the invoking and/or invoked protection domains. This most commonly occurs when a thread wishes to map, for the duration of the portal invocation, a region of memory belonging to the invoking protection domain into the virtual address space of the invoked protection domain; this gives the thread a window into the address space of the invoking protection domain while running in the invoked protection domain. When the thread returns, the window is closed.

Such a memory window can be used to save the cost of copying data between protection domains. Variations include windows that remain open (to share pages between protection domains), windows that transfer pages from the invoking domain to the invoked domain (to implement tear-away write) and windows that transfer pages from the invoked domain to the invoker (to implement tear-away read)

Note that although the portal code may modify VM data structures, only the VM manager and the portal manager (which generates portal code) share the knowledge about these data structures. The Pebble nucleus itself is oblivious to those data structures.

Portal code may never block and may not contain loops. This is essential to ensure that the portal can be traversed in a small, finite amount of time. If the portal has to block (e.g. the invoked domain's stacks queue is empty), then the portal code transfers control to the scheduler, inside which the calling thread is waiting for the resource.

An important point that has not yet been mentioned is that specialized portal code is generated, on-the-fly,

when a portal is opened. This allows portal code to take advantage of the semantics and trust relationships of the portal. For example, if the caller trusts the callee, the caller may allow the callee to use the caller's stack, rather than allocate a new one. If this level of trust does not exist, the caller can require that the callee allocate a new stack. Although sharing stacks decreases the level of isolation between the caller and callee, it can improve performance.

4.2 Server Components

As part of the Pebble philosophy, system services are provided by *server components*, which run in user mode inside separate protection domains. Unlike applications, server components may be granted limited privileges not afforded to application components. For example, the scheduler runs with interrupts disabled, device drivers have device registers mapped into their memory region, and the portal manager may add portals to protection domains (a protection domain can not modify its portal table directly).

There are many advantages for implementing services at user level. First, from a software engineering standpoint, we are guaranteed that a server component will use only the exported interface of other components. Second, because each server component is only given the privileges that it needs to do its job, a programming error in one component will not directly affect other components. Clearly, if a critical component fails (e.g., VM) the system as a whole will be affected—but a bug in console device driver will not overwrite page tables.

4.3 Scheduling and Synchronization

Pebble's scheduler implements all actions that may change the calling thread's state (e.g. $run \rightarrow blocked$ or $blocked \rightarrow ready$). Threads cannot block anywhere except inside the scheduler. In particular, Pebble's synchronization primitives are managed entirely by the user-level scheduler. When a thread running in a protection domain creates a semaphore, two portals that invoke the scheduler (for *P* and *V* operations) are added to the protection domain's portal table. The thread invokes *P* in order to acquire the semaphore. If the *P* succeeds, the scheduler grants the calling protection domain the semaphore and returns. If the semaphore is held by another protection domain, the *P* fails, the scheduler marks the thread as blocked, and then schedules another thread. A *V* operation works analogously; if the operation unblocks a thread that has higher priority than the invoker, the scheduler can block the invoking thread and run the newly-awakened one.

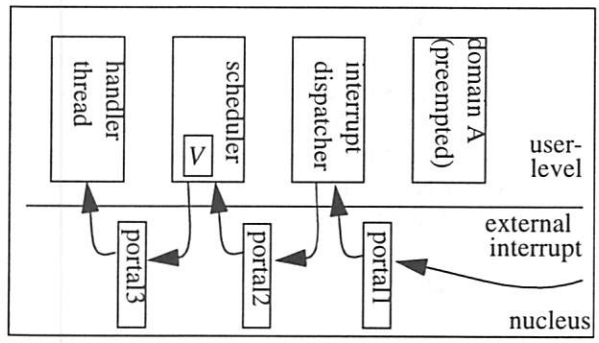

Figure 2. Interrupt Handling. An interrupt causes a portal call to the interrupt dispatcher, which calls the scheduler to performs a *V* operation on the device's semaphore. The scheduler wakes up the handler thread that waits on this semaphore. See text for explanation of different portal activities.

The scheduler runs with interrupts disabled, in order to simplify its implementation. Work on the use of lock-free data structures has shown that, with appropriate hardware support, it is possible to implement the data structures used by Pebble's scheduling and synchronization component without locking [Herlihy91]. Such an implementation would allow the scheduler to run with interrupts enabled which would reduce interrupt latency even further. (We have not yet implemented the scheduler using such data structures, although we plan to investigate such an implementation in the future.)

4.4 Device Drivers and Interrupt Handling

Figure 2 depicts the interrupt handling in Pebble. Each hardware device in the system is associated with a semaphore that used to communicate between the interrupt dispatcher component and the device driver component for the specific device. Typically, the device driver will have left a thread blocked on that semaphore.

The portal table of each protection domain contains entries for the machine's hardware interrupts. When an interrupt occurs, *portal1* saves the context of the currently running thread, including the contents of the entire register set. *Portal1* then switches the stack pointer to the interrupt stack and calls the interrupt dispatcher, which determines which device generated the interrupt. The interrupt dispatcher calls the scheduler to performs a *V* operation on the device's semaphore via *portal2*. This portal saves only a few registers and allows the scheduler to share the same stack as the interrupt dispatcher. The *V* operation unblocks the handler thread. If the handler thread has a higher priority than the thread which was running at the time when the inter-

rupt was received, the scheduler calls *portal3* with the identity of the handler thread. *Portal3* restores the context of the handler thread, including registers and stack, and the interrupt is handled immediately. Otherwise, the handler thread is added to the ready queue and the scheduler selects to resume the thread which was running previously by calling *portal3* with the identity of this thread. Note that *portal3* performs the actual context switch. The scheduler just supplies the identity of the next thread to run.

Note that Pebble does not rely on hardware interrupt priorities in order to schedule interrupt handler threads. The interrupt dispatcher is called promptly for all interrupts, and the Pebble scheduler decides whether to run the associated handler thread. Pebble unifies interrupt priority with thread priority, and handles both in the scheduler

4.5 Writing Portal Code

Portal definitions are written using a simple interface definition language. The portal manager uses this definition to dynamically generate specialized code when a portal is created. The interface definition specifies which registers to save, whether to share a stack with the receiving domain, and how to process each argument.

Simple arguments (e.g., integers) are not processed at all; more complex argument types may require more work. For example, an argument may specify the address of a memory window that should be mapped into the receiver's address space, or a capability (see section 4.7) that must be transformed before being transferred. On one hand, in order to be efficient, such transformation code may need to have access to private data structures of a trusted server (e.g., the virtual memory system or the capability system); on the other hand, the trusted servers should be allowed to keep their internal data representations private.

The solution we plan to implement is to allow trusted services to register argument transformation code templates with the portal manager. When the portal manager instantiates a portal that uses such an argument, the code template is used when generating the portal. This technique allows portal code to be both efficient (by inlining code that transforms arguments) and encapsulated (by allowing servers to keep their internal representations private). Although portal code that runs in kernel mode has access to server-specific data structures, these data structures cannot be accessed by other servers.

4.6 Short-Circuit Portals

In some cases the amount of work done is so small that the portal code itself can implement the service. A *short-circuit* portal is one that does not actually transfer the invoking thread to a new protection domain, but instead performs the requested action inline, in the portal code. Examples include simple "system calls" to get the current thread's ID and obtain the time of day. The TLB miss handler (which is in software on the MIPS architecture, the current platform for Pebble) is also implemented as a short-circuit portal.

4.7 Capabilities

Pebble's design includes support for *capabilities*, abstract tokens that represent access rights. The capability manager, a trusted user-level server, keeps track of the capabilities available to each protection domain. Additionally, it registers a capability argument type and associated transformation code with the portal manager. When a capability is passed through a portal, the portal code adds the capability to the receiving protection domain's capability list, and transforms the sending protection domain's external representation of the capability to that of the receiving domain. We plan to implement the standard capability operations (e.g., revocation, use-once, non-transferability, reduction in strength, etc.).

5 Portal Example

Portals can be used to model not only code, but also data; a set of portals can be used to represent an open file descriptor. In Pebble, an *open* call creates three portals in the invoking protection domain, one each for *read*, *write*, and *seek* on the corresponding file. A *read* call transfers directly to the appropriate routine, so no run-time demultiplexing is needed to determine the type of the underlying object; the appropriate code (for a disk file, socket, etc.) will be invoked. Additionally, a pointer to its control block can be embedded in the portal code and passed directly to the service routine, so no run-time validation of the file pointer needs to be done. Because the portal code can not be modified by the client, the server can trust that the control block pointer passed to it is valid, and can access the particular file immediately. There is no need for a separate file descriptor table; the data normally associated with the tables is found in the dynamically generated portal code.

6 Performance

We measured the operation of Pebble for several micro-benchmarks on three different test hardware platforms, named LOW, MID and HIGH, which represent low-, medium- and high-end embedded system configurations, as described in Table 1. All three platforms included MIPS processors from QED (RM5230 and RM7000) and IDT (R5000). All motherboards were manufactured by Algorithmics, a developer of systems for embedded applications.

The LOW platform is representative of low-cost hand-held devices, which have a single-level cache hierarchy and smaller memory. The MID platform is representative of more powerful appliances, such as a set-top box, which contain a more powerful processor, two-level cache hierarchy and larger memory. The HIGH platform is representative of high-end systems, which contain top-of-the-line processors with larger caches (Note that it is inevitable that over time the HIGH platform will come to be considered MID and the MID platform LOW.).

The L1 cache in all targets can be accessed in a single machine cycle, and does not cause any pipeline delay. Access to higher levels of the memory hierarchy causes a delay, which is depicted in Table 1.

6.1 Basic Operations

Table 2 depicts the time to perform several primitive operations on the three test platforms. We present the results in machine cycles and not in absolute time, since we believe that these measurements will remain the same (in machine cycles) for faster processors with similar memory hierarchy. The reported times are the average of 10,000 repetitions.

All of the operations reported in Table 2 are simple communication or synchronization building blocks. Their performance suggests that higher-level operations will be efficient as well.

The reported operations are:

- **s-c portal:** the time of the `get_asid()` short-circuit portal, which returns the identity of the calling domain. This is the equivalent of the UNIX null system call. A short-circuit portal performs the action and returns to the caller immediately without a context switch.

	LOW	MID	HIGH
board	P4032	P5064	P5064
processor	RM5230	R5000	RM7000
processor speed (MHz)	133 Mhz	166 MHz	200 MHz
pipeline	single	single	dual
L1 cache	2-way 16 KB I + 16 KB D	2-way 32 KB I + 32 KB D	4-way 16 KB I + 16 KB D
L2 cache	—	off-chip direct-map 1 MB	on-chip 4-way 256 KB
L2 access (cycles)	—	10	3
L3 cache	—	—	off-chip direct map 2 MB
L3 access (cycles)	—	—	17
main memory	16 MB	64 MB	64 MB
memory access (cycles)	40	26	41

Table 1. Test hardware. We ran our tests on three platforms that represent three points in the embedded system hardware spectrum. The platforms share a common CPU architecture, and vary in cache size and architecture and processor speed.

- **ipc:**[1] the time for one leg of an IPC between two domains, which is implemented by a portal traversal. The portal passes four integer parameters in the machine registers. No parameter manipulation is performed by the portal code. The portal allocates a new stack in the target domain and frees the stack on return. The reported time in Table 2 is one leg time (half of the total time). Additional measurements

1. The acronym *ipc* traditionally refers to an *inter-process communication*. By serendipity, we can also use *ipc* to represent an *inter-protection domain call*; in Pebble, the two are equivalent.

(reported in [Gabber99]) show that the per-leg time is constant for a chain of IPCs through a sequence of domains.

- **yield:** measures the thread yield operation, in which the current thread calls the scheduler and requests to run the next thread with a higher or same priority. We present four measurements, for one, two, four and eight threads (**yield1**, **yield2**, **yield4**, and **yield8**, respectively). There is one active thread in each domain. The reported time in the table is a single context switch time (total time / total number of yields by all threads).
- **sem:** measures the time to pass a token around a ring of n threads, each running in a separate domain. Each thread shares one semaphore with its left neighbor and one semaphore with its right neighbour. The thread waits on its left neighbor. Once this semaphore is released, the thread releases its right semaphore and repeats the process. We present four measurements, for one, two, four and eight threads (**sem1**, **sem2**, **sem4**, and **sem8**, respectively). There is one active thread in each domain. The reported time is the time to pass the token once, which is the time for a single pair of semaphore acquire/release.

The results reported in Table 2 indicate that the performance degrades with the number of active threads, which is expected due to more frequent cache misses. Performance degrades most with LOW platform, since it has no L2 cache, and least with HIGH platform, which has a large L3 cache that is large enough to hold the entire working set of the test.

Note that the HIGH platform is not significantly faster than the LOW and MID platforms for **s-c portal**, **ipc** and **sem1** tests, although it has a dual-issue pipeline and much larger caches. This is because these tests do not cause much L1 cache misses, and the portal code is dominated by sequences of load and store instructions that cannot be executed in parallel.

6.2 Interrupt Latency

When an interrupt is generated, there are two factors that control the delay before the interrupt is handled. First, there can be a delay before the system is notified that an interrupt has been received, if the processor is running with interrupts disabled. Second, there may be a delay between the time the interrupt is delivered and the time the device is serviced. The sum of these two delay components is the interrupt latency.

operation	LOW	MID	HIGH
s-c portal	44	45	41
ipc	118	117	119
yield1	667	423	348
yield2	661	425	346
yield4	1593	549	346
yield8	1628	549	452
sem1	543	549	524
sem2	775	781	720
sem4	1872	942	720
sem8	1878	1198	857

Table 2. Basic operations time (cycles). These measurements give an estimate of the performance of Pebble on basic communication operations.

The first delay component, L, is bounded (below) by the length of path through the system where interrupts are disabled.[1] In particular, L is determined by the portal code and by the scheduler, which are the only frequently-used portions of the system that run with interrupts disabled.

The value of L is the time reported in Table 2 for **s-c portal**, **ipc** and **yield1-yield8**. It is half the time reported for **sem1-sem8**, since these times are for a pair of semaphore operations.

The second delay component, C, is bounded (below) by the minimum amount of time required to deliver the interrupt to the interrupt handler. Thus the interrupt latency will range from $[C, C+L]$, provided that interrupt handling does not generate many additional cache misses, such as in the MID and HIGH platforms.

Table 3 shows the interrupt response time of Pebble when the system is performing different background tasks. We present the median and 99th percentile response time of 10,000 repetitions.

The interrupt latency is measured by computing the difference between the time that an interrupt was generated

1. Length, in this context, refers to the amount of time it takes to process the instruction sequence. Intuitively, it can be expected to be roughly proportional to the length, in instructions, of the code path.

and the time that a user thread that waited for this interrupt actually woke up. To accomplish this, we have the measurement thread sleep for some (randomly chosen) duration, and compare the time it is woken up with the time it expected to be woken up. The difference between the two is the interrupt latency.

This test is very precise; each of the test platforms include a high-resolution timer that is incremented every second processor cycle and the ability to schedule timer interrupts with the same granularity. The code fragment Figure 3 shows the inner loop of the measurement thread.

```
for (i = 0; i < N; i++) {
  delay = rand(MIN, MAX);
  start = hrtime();
  hrsleep(delay);
  latency = hrtime() - start - delay;
}
```

Figure 3. Measurement Thread. The code of the inner loop of the interrupt latency measurement thread.

The `rand()` function generates a uniformly distributed random number in the range [MIN, MAX]. The `hrtime()` function returns the current high-resolution time. The `hrsleep()` routine waits until the specified time.

In order to estimate interrupt latencies under various loads, we run the measurement thread concurrently with a background task that repeatedly performs a specific operation. Different operations exercise different interrupt-disabled paths of the system, and hence will have different interrupt latency characteristics. The background threads we tested were:

- **idle:** a background thread that spins in a tight loop on a particular variable in user mode. The idle task can be preempted at any time. The value reported for this case is an estimate of *C*, the lower bound of the interrupt latency.
- **s-c portal**: a background task that calls the `get_asid()` routine repetitively. `Get_asid()` is implemented by a short-circuit portal. The background thread is identical to the **s-c portal** program from Table 2. Interrupts are disabled while executing this portal.
- **ipc**: a background thread that repeatedly performs an IPC to the protection domain in which the measurement thread is running. The portal code associated with this portal is minimal, just transferring control, and call itself returns immediately. The background thread is identical to the **ipc** program from Table 2. Interrupts are disabled during each leg of the IPC (call and return), and are enabled when executing in the caller and called domains.
- **yield**: a background thread that repeatedly calls the scheduler to yield control. This thread is identical to the **yield1** program from Table 2. As there is one active thread in the system, the scheduler just returns to the calling thread. Interrupts are disabled during the portal call to the scheduler, inside the scheduler, and during the portal return to the thread.
- **sem**: a pair of background threads, which pass a token back and forth. Each thread runs in separate protection domain. This test is identical to the **sem2** program from Table 2. Interrupts are disabled during the semaphore operations.

background		LOW	MID	HIGH
idle	median	1200	1294	1224
	99th %	1298	1296	1228
	max.	2164	1322	1322
s-c portal	median	1170	1176	1240
	99th %	1240	1200	1262
	max.	2026	1202	1266
ipc	median	1152	1224	1274
	99th %	1240	1290	1340
	max.	2088	1300	1342
yield	median	1210	1346	1404
	99th %	1474	1580	1584
	max.	2282	1584	1588
sem	median	2302	1492	1400
	99th %	3024	1712	1596
	max.	3996	1722	1604
est. interrupt latency lower bound (*C*)		9.0µs	7.8µs	6.1µs

Table 3. Interrupt latency (cycles). The time, in cycles, between the expected delivery of an interrupt and when it is received.

Table 3 indicates that the interrupt latency is bounded by the sum of a platform-specific constant plus a time

which is proportional to the longest interrupt-disabled path in the background task. The platform-specific constant is the minimal interrupt response time *(C)*, which is the median value of the **idle** test. The measurements in Table 2 are the upper bound of the duration of the corresponding interrupt-disabled paths.

Note that the median and the 99th percentile of the **idle** test on all platforms are very close. This indicates that the interrupts are served immediately.

The maximal interrupt latency on the MID and HIGH platforms is very close to the 99th percentile on these platforms, which indicates that the system performance is highly predictable. However, the maximal interrupt latency on the LOW platform is up to 60% higher than than the 99% percentile latency. This is the result of the small cache size of LOW, which result in excessive cache misses due to infrequent background events, such as the timer interrupt.

In the case of the LOW and MID platforms, the **s-c portal** and **ipc** tests had a slightly lower median interrupt latency than the **idle** test, but they differ by less than 5%. This may be caused by a better cache hit rate.

We see that the interrupt latencies for the MID and HIGH systems are quite close, and much lower than those for LOW. Although the cache architectures of the two differ, both MID and HIGH appear to have more effective caches than LOW. The L1 cache of MID is twice the size of the L1 cache of LOW, and although the L1 cache of HIGH is the same size as that of LOW, it is 4-way set associative. In addition, the L2 cache of HIGH can be accessed in only three cycles. Thus an L1 miss on HIGH is not as painful as it would be on MID or LOW.

6.3 Summary

In this section we have shown microbenchmark results for Pebble that indicate that the cost of hardware protection of both components and system services is very low (as summarized in Table 2) and that interrupt latency on Pebble is quite low, with a lower bound of 1200-1300 cycles (6.1µs to 9.0µs) depending on the target architecture.

7 Related Work

Traditional microkernels, such as Mach, Chorus, and Windows NT leave much more of the operating system's code running in privileged mode than does Pebble. In fact, the trend has been to run more of the system in privileged mode as time goes by—the current release of NT includes the window system in the privileged mode kernel.

By moving services into the privileged mode kernel to reduce communication overhead, operating system designers are, in essence, giving up on the microkernel approach. Recent work has focused on finding and fixing the performance bottlenecks of microkernel approach, which has required rethinking its basic architecture.

Liedtke, in his work on L4, has espoused the philosophy of a minimal privileged mode kernel that includes only support for IPC and key VM primitives [Liedtke97]. Pebble goes one step further than L4, removing VM as well (except for TLB fault handling, which is done in software on MIPS).

The Exokernel [Kaashoek97] attempts to "exterminate all OS abstractions," leaving the privileged mode kernel in charge of protecting resources, but leaving abstraction of resources to user level application code. As with the Exokernel approach, Pebble moves the implementation of operating system abstractions to user level, but instead of leaving the development of OS abstractions to application writers, Pebble provides a set of OS abstractions, implemented by user level OS components. Pebble OS components can be added or replaced, allowing alternate OS abstractions to coexist or override the default set.

Pebble was inspired by the SPACE project [Probert91], which was in turn inspired by the Clouds project [Dasgupta92]. Many of the concepts and much of the terminology of the project come from these systems; e.g., SPACE provided us with the idea of cross-domain communication as a generalization of interrupt handling.

Pebble applies techniques developed by Bershad et al. [Bershad89], Massalin [Massalin92], and Pu et al. [Pu95] to improve the performance of IPC. Bershad's results showed that IPC data size tends to be very small (which fits into registers) or large (which is passed by sharing memory pages). Massalin's work on the Synthesis project, and, more recently, work by Pu et al. on the Synthetix project, studied the use of generating specialized code to improve performance.

8 Status

The Pebble nucleus and a small set of services (scheduler, portal manager, interrupt dispatcher, and minimal

VM) and devices (console, clock, and ethernet driver) currently runs on the MIPS processor. Work on implementing networking support (TCP/IP), file system support, and a port to the x86 is underway.

9 Summary

The Pebble architecture provides for an efficient operating system that is easy to modify and debug. Pebble provides hardware protection for running components written in any language. Components communicate via portals and run in user mode with interrupts enabled. Through the use of portals, which are specialized to the specific interrupt or communication they are to handle, Pebble is able to "compile out" run-time decisions and lead to better performance than typical operating system implementations. In addition, a portal can be configured to save and restore only a subset of the machine state, depending on the calling conventions of, and the level of trust between, the client and server. Additionally, the low interrupt latency provided by the Pebble architecture makes it well-suited for embedded applications.

10 References

[Bershad89] B. Bershad et al., "Lightweight Remote Procedure Call," *Proc. 12th SOSP*, pp. 102–113 (December 1989).

[Dasgupta92] P. Dasgupta, R. LeBlanc, M. Ahamad, U. Ramachandran, "The Clouds Distributed Operating System," *IEEE Computer 24*, 11 (November 1992).

[Dorward97] S. Dorward, R. Pike, D. Presotto, D. Ritchie, H. Trickey, P. Winterbottom, "Inferno," *Proc. IEEE Compcon 97*, pp. 241–244 (1997).

[Gabber99] E. Gabber, J. Bruno, J. Brustoloni, A. Silberschatz, C. Small, "The Pebble component-based operating system," *Proc. 1999 USENIX Technical Conference*, Monterey, CA (June 1999).

[Gosling96] J. Gosling, B. Joy, G. Steele, *The Java™ Language Specification*, Addison-Wesley, Reading, MA (1996).

[Herlihy91] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems 13*, 1 (January 1991)).

[Kaashoek97] M. F. Kaashoek et al., "Application Performance and Flexibility on Exokernel Systems," *Proc. 16th SOSP*, pp. 52–65 (October 1987).

[Liedtke97] J. Liedtke et al., "Achieved IPC Performance," *Proc. 6th HotOS*, pp. 28–31 (May 1998).

[Massalin92] H. Massalin, *Synthesis: An Efficient Implementation of Fundamental Operating System Services*, Ph.D. thesis, Columbia University (1992).

[Mendelsohn97] N. Mendelsohn, "Operating Systems for Component Software Environments," *Proc. 6th HotOS*, pp. 49–54 (May 1978).

[Probert91] D. Probert, J. Bruno, M. Karaorman, "SPACE: A New Approach to Operating System Abstractions," *Proc. IWOOOS*, pp. 133–137 (October 1991). Also available from ftp.cs.ucsb.edu/pub/papers/space/iwooos91.ps.gz

[Pu95] C. Pu et al, "Optimistic Incremental Specialization: Streamlining a Commercial Operating System," *Proc. 15th SOSP*, pp. 314–324 (December 1995).

Massively Distributed Systems: Design Issues and Challenges

Dan Nessett
Technology Development Center
3Com Corporation
Dan_Nessett@3com.com

Abstract

The forty year trend in the computing industry is away from centralized, high unit cost, low unit volume products toward distributed, low unit cost, high unit volume products. The next step in this process is the emergence of massively distributed systems. These systems will penetrate even more deeply into the fabric of society and become the information power grids of the 21st century. They will be ubiquitous. Most will operate outside the normal cognizance of the people they serve and most will be based on embedded systems that present non-traditional computing interfaces to their users. They will be engineered to operate as distributed utilities, much like the energy, water, transportation and media broadcast businesses do today. The first deployment of massively distributed systems is likely to occur as support structures for these industries.

Massively distributed systems will differ from existing distributed systems in important ways. Such systems eventually will interconnect billions of nodes. This will necessitate changes in the way nodes interact with one another. One-to-many communications will be the norm, rather than one-to-one. The size of on-line application communities will necessitate the use of statistically correct rather than deterministic algorithms for resource accounting, fault detection and correction, and system management. These communities will coalesce and dissolve rapidly in order to host events that are of interest to groups formed specifically for this purpose. This will require new approaches to naming, routing, security and privacy, resource management and synchronization. Heterogeneity will be even more of a factor in the design, implementation and operation of massively distributed systems

This paper explores the nature and characteristics of massively distributed systems by proposing some examples and then using them to characterize nine major design issues. Seven of these are drawn from seminal work in the area of distributed systems. Two others are based on experience in distributed system design and implementation subsequent to that work.

1. Introduction

This paper explores trends in distributed system architecture and implementation that will have far-reaching consequences in the next 5-10 years. The paper's central thesis is that applications will become massively distributed during this interval and in so doing generate significant engineering problems, the solution of which will change the nature of distributed computing. Following this trend, networking technology will change in order better to serve the communications requirements of this new class of applications.

One major driver of massively distributed applications is the advent of ubiquitous embedded systems. While embedded systems will not always communicate either with their peers or with traditional computing equipment, important applications will require the interconnection of them in large numbers. Several examples are given in section 2. Furthermore, these applications will require computing architectures that off-load heavy computational tasks from embedded systems onto more powerful and capable equipment. Consequently, massively distributed applications will present problems to architects and implementers that transcend those of stand-alone embedded system applications.

The paper examines the motivation for and engineering problems of massively distributed systems. In the next section, a case is made why massively distributed systems will arise. Section 3 presents an analysis of their engineering design issues. Finally, section 4 presents a summary of the paper.

2. Motivation

Massively distributed systems are the next step in the development of computing. From a commercial perspective, this process started with centralized, high unit cost, low unit volume systems in the fifties and sixties. It then moved to less centralized, but still more or less stand-alone systems in the seventies. In the eighties, Network Service Providers (NSPs), such as Compuserv, Genie and America On-line, became popular providing email and bulletin board services to businesses and some advanced home users. In the first half of this dec-

ade, internet access became popular, supporting truly distributed applications, such as the World Wide Web. In the first decade of the 21st century, massively distributed systems will appear and become ubiquitous. In direct contrast to the systems of the fifties through the early nineties, these systems will be decentralized, low unit cost and high unit volume. They will be pervasive. Most will operate outside the normal cognizance of the people they serve and most will be based on embedded systems that present non-traditional computing interfaces to their users. They will be engineered to operate as distributed utilities, much like the energy, water, transportation and media broadcast businesses do today. Indeed, the first deployment of massively distributed systems is likely to occur as support structures for these industries.

Massively distributed systems will not be limited in scope to the interconnection of and interaction between embedded systems alone. While large numbers of embedded systems will participate in massively distributed systems, at least some will interact with traditional computing equipment. The latter will provide control, information storage, intensive computation and other services to massively distributed applications. Embedded systems will provide specialized point-of-presence services.

Several hypothetical examples illustrate the future impact of massively distributed systems. These will arise as natural extensions to current practice. They will be used subsequently to illustrate some of the distinguishing characteristics of massively distributed systems, which are significantly different from those of smaller scale distributed systems operating today.

2.1. Ubiquitous supply-chain management

Managing large distribution processes in the face of increasing governmental regulation, resource scarcity and system complexity will become increasingly important and open up markets for massively distributed systems. The classic utility businesses, e.g., the electrical, water, natural gas, and telephone companies, will be joined by other flow based service industries, e.g., transportation conglomerates, oil companies (especially those which own and operate service stations), large retail companies, including food, clothing, soft and hard goods businesses, and the information industries, such as newspaper, television, entertainment and advertisement concerns to concentrate on improving their margins by the efficient control of their product flow. Supply-chain management will spread from manufacturing to the day-to-day operations of companies close to the consumer. This will require the development and deployment of massively distributed systems that reach into the home, office, public pathways, and private institutions.

While the problems presented by these massively distributed applications will display certain differences, there will be a surprising amount of commonality. Managing flows, whether they consist of information, paper, dishwashing detergent, gasoline, or other commodities, requires sensors at the points where the product is consumed, quickly configurable transportation systems to move those commodities (e.g., trucks, power switching systems, airplanes, a communications fabric), and control algorithms and equipment to match the ingress of raw materials to the egress of the consumed products.

In many distributed systems, such as those concerned with power, water, natural gas, vehicle fuel, and mass market goods delivery, embedded processors will measure resource use and relay consumption information up the supply-chain. Control systems will utilize this information to schedule raw material inventory replenishment and organize the management of resource flows. Embedded systems close to the consumer (e.g., in appliances, yard sprinkler systems, home heating systems) will schedule resource consumption to minimize resource demand spikes, thereby reducing the peak capacity requirements of delivery systems [1].

An important supply chain that will dramatically influence how massively distributed systems are implemented is information flow. The resource management systems mentioned in the previous paragraph are geographically hierarchical, since they control physical processes characterized by the movement of mass or energy. Information flows as easily over large as small distances, a characteristic that distinguishes it from these other supply chains. Consequently, information supply chains will present significantly different problems to designers and implementers of massively distributed systems. Embedded systems will play a large role in information supply chains, as discussed in the next section.

It will be necessary to use massively distributed systems to solve this new class of flow control problems, since the ratio of control points to production facilities will grow to be several orders of magnitude larger than it is today. This adjustment will encourage new control algorithms based not on deterministic concepts, but rather on statistical notions of correctness. Profits will be maximized by ensuring a less than 100% accounting of resource usage and payment, since reaching the last

few percent will impose significant marginal costs that will exceed the marginal return. Banks already employ this philosophy in response to ATM fraud. The money that could be used to make their equipment more secure earns more interest than the losses they would prevent. Of course, to ensure an equitable and legally defensible operation (not to mention keeping stockholders happy), control algorithms must eliminate systematic fraud whereby one or a small group of individuals consistently benefit from a system's non-deterministic behavior.

2.2. Agile distributed information services

Perhaps the most dramatic development of the past 5 years is the growth of information distribution channels based on the Internet and their rapid penetration into areas not normally associated with computing. The World Wide Web, email, and Internet news groups are prominent examples. This trend will not only continue it will accelerate. The maturation of multi-media broadcast technology running over the Internet, assisted by extremely high capacity networking infrastructure will encourage the formation of virtual information communities. Unlike more traditional groupings, these communities will be ephemeral, lasting anywhere from weeks or months to periods as short as several hours or less. Primitive examples of these communities now exist, e.g., everyone reading the same newspaper and all those watching the same TV program. However, the information communities of tomorrow will not be based on rigidly structured broadcast hierarchies; rather they will encourage interactions between people who are members of a community at a particular moment. Multicast interest groups centered around a wide variety of subjects will form, much as internet news groups form today. Small production companies will present a play, host a debate, cover a sporting event or sponsor an electronic interactive event, such as a networked game that millions play simultaneously. The large media conglomerates will gradually evolve into distributed information and entertainment production companies offering a wider variety of programming material than they do now.

Managing such rapidly changing information communities will require more capable networks than currently exist. Switching and routing equipment will include embedded systems to monitor resource use and adjust allocations to meet fluctuating demand. These systems will control resources that are traditionally static, such as device backplane bandwidth, encryption and compression hardware, and agile error detection/correction hardware. Embedded processors will run transportable code, such as Java, in order to support active networking [2].

To protect consumers from unwanted intrusions into their lives, products will be developed to filter the information flowing to an end-system. These filters will range from those that allow parents to control the information available to their children (eliminating offensive material such as pornography) to those that eliminate the probable increase in junk information broadcast to consumers. Specialized processors implementing pattern recognition algorithms and embedded in routing, switching and end systems will provide powerful and customizable filtering capabilities to achieve these objectives.

To solve the problem of locating communities that might interest an individual, services will form offering electronic real-time catalogues. These will be the successors of the WWW index and search engines that exist currently. Electronic advisory services will inform consumers which communities offer the best value, using a value metric tailored for the customer's specific objectives. Specialized processors embedded in database and information repository systems will continuously index and catalogue information as it arrives.

Needless to say, participation in these communities will require payment, ranging from that necessary simply to reimburse common carriers, when the content is free (e.g., broadcasts of school plays, candidate sponsored political events, religious events), to payments for exclusive information commodities (e.g., pay-per-view sporting events, specialized financial broadcasts). Such payments will require the development and deployment of an electronic monetary system that utilizes a combination of digital cash, electronic credit systems (i.e., electronic analogues of the credit card, bank line of credit, and checks), and point of presence payment terminals. Processors embedded in cash cards and other payment tokens will play an important role in such systems.

2.3. The new economics

Electronic commerce is now a common feature of many commercial enterprises. However, most designers and implementers of electronic commerce systems have concentrated on the provision of electronic payment. Other aspects of electronic commerce have received less attention, in particular, many components that are well matched to implementation on massively distributed systems. Commerce involves not only payment for goods and services, but also their creation, advertisement, delivery, maintenance, and disposal.

The downward spiraling cost of communication is changing profit models in many businesses. Hard product companies will change their businesses to adapt to the new micro-economic environment. The creation of hard goods will become increasingly automated. Their advertisement will become more interactive, consumers will use third party search and evaluation companies that recommend products according to a detailed specification. Since consumers are rarely expert enough to create a useful and detailed description of their requirements, such companies will develop interactive systems with friendly customer interfaces that will lead a customer in the development of these requirements.

Customers will not travel to a remote location to use these systems. They will be accessed through communications facilities terminated in the customer's home or place of business. Eventually, requirement specifications will drive the creation of the product at an automated facility, create a just-in-time delivery plan to move the product from the manufacturing facility to the shipping end-point, schedule the resources to conduct warranted and purchased maintenance on the product, and arrange for the disposal of old equipment the customer no longer needs (or that was traded in for new equipment). Companies will optimize costs by coordinating this with the warehousing and shipment of the purchased equipment as well as with the disposal of the equipment of other customers.

Embedded systems will play an important role in the new economics. Once a manufacturing facility receives a product specification, embedded processors in robotic equipment will tailor its mechanical assets to create that product. Embedded processors will sense resource usage and work with control systems to ensure parts are delivered when necessary. Embedded systems will monitor and schedule warehouse and delivery capacity, cooperating with control systems to move the product to the customer. Embedded systems in robots will manage the repair of products returned for maintenance.

3. Design issues

Massively distributed systems are not only quantitatively different, but also qualitatively different than the distributed systems in place today. Their size and scope introduce new problems in design and implementation. However, to a certain extent we can use the experience we have gained building moderately scaled distributed systems to guide us in solving these problems.

This section presents a taxonomy of design issues for massively distributed systems. It is based on seminal work [3] that analyzed the current generation of distributed systems. This work identified seven major design issues. Two additional issues are added to the seven previously identified based on experience subsequent to that work. The classical design issues are : 1) naming, 2) error control, 3) resource management, 4) protection (security and privacy), 5) synchronization, 6) objects (representation, encoding and translation), and 7) measurement, testing and debugging. The additional design issues are : 8) heterogeneity, and 9) scale.

Each of these issues is used to show how massively distributed systems differ significantly from existing distributed systems. For pedagogical reasons, the issues are addressed in the following order : scale, heterogeneity, objects (representation, encoding and translation), resource management, protection (security and privacy), naming, error control, synchronization, and measurement, testing and debugging.

3.1. Scale

The main characteristic of massively distributed systems is their scale. While current distributed system are composed of hundreds of nodes[1], during the first decade of the 21st century massively distributed systems will be composed of thousands to billions of nodes, supporting hundreds to millions of users [4].

The scaling issue for massively distributed systems has a number of dimensions. In the area of communications capacity, recent events in the telecommunications industry promise to dramatically change the available worldwide bandwidth for networking. For example, Project Oxygen of the CTR Group, Ltd. intends to lay fiber across the ocean floors resulting in bandwidth of 1.28 terabits/sec on any segment [5]. Current schedules call for the completion of an Atlantic ring by Q4 2000 and a Pacific ring by Q2 2001. The pricing schedules proposed by CTR Group will result in transoceanic bandwidth offered at 1.5% of current satellite circuit prices and 1% of marine cable prices.

Power, water, natural gas, vehicle fuel and mass market supply chains are naturally large scale. Traditional information supply chains (e.g., radio, television, motion picture, the printed news media) are broadcast based and characterized by a small producer-to-consumer ratio. However, this is undergoing radical change. The growth of the world wide web has greatly increased the pro-

[1] Some may object to this assertion, observing that the current Internet is composed of millions of nodes. However, the term distributed system here means a cohesive set of computing resources acting together to achieve a common objective. While the routing infrastructure of the Internet can be considered to be a massively distributed application, no other system of this scale exists currently.

ducer-to-consumer ratio in data communications and this trend will continue. There may even come a time when the ratio approaches 1. This one factor has the potential to completely change how information supply chains are implemented and could be the major driver behind massively distributed systems.

Geographically, massively distributed applications supporting information supply chains will be global in extent. A single application running over a massively distributed system will interconnect users of significantly different languages, cultures and views. Communication costs will be structured so that they scale logarithmically with the number of destinations, thereby utilizing efficiencies inherent in broadcast and multicast communications. Individual subscribers will be able affordably to communicate with one to a hundred other subscribers. One-to-many communications will be the norm, as opposed to one-to-one transfers, which are most common today.

3.2. Heterogeneity

While heterogeneity is an important design issue for existing distributed systems, its extent in massively distributed systems will be significantly greater. In particular:

- The communications infrastructure will be composed of channels of very different capacities. Very low bandwidth channels (e.g., 1200 BPS) will still be in use in developing nations during the next decade, while the developed world will support very high bandwidth channels in the core services (e.g., 1 Tbps). Communications between embedded systems and their peers or higher-level facilities will occur over channels of significantly lower bandwidth than that available in the core of the network. Building applications that run over bandwidth diverse communication infrastructure will require new ways of organizing data flows.

- Similarly, end-systems will possess a wide variety of presentation techniques, from interactive HDTV to personal digital assistants and personal phone equipment. Applications will combine these end-systems into coherent interactive subsystems. In fact, it is possible today to create a communications conferencing system that patches together video-teleconferencing equipment, multicast capable workstations and cellular phones.

- Participatory communities, formed on demand, will choose from an array of costing options for their members. Short-lived communities may elect a pay-per-use approach, which is suited to their ephemeral nature. Longer-lived communities may be more inclined to charge a subscription fee for their services. Some communities with external financial backing may choose to charge no fee, transferring value to their sponsors through advertising, indirect persuasion, or charitable benefit. Each of these costing models must be supported by the massively distributed system infrastructure in a way that allows such diverse activities to exist concurrently.

- To support cost recovery for distributed applications, several efforts are underway to build and deploy electronic monetary systems. Existing applications are able to use a single funds transfer system, since many serve a limited customer base. Massively distributed systems, on the other hand, will be global in scope and normally serve customers in many countries who use a large variety of payment systems. For some, only traditional methods, such as a credit card or cash payment card will be available; while others may use different electronic systems, such as digital cash, digital checks, or digital credit cards. Embedded systems will play an important role in the latter class of payment system. Massively distributed systems must accommodate the use of all such systems by a single application. This will necessitate the development of funds transfer transaction clearinghouses that will convert funds from one system to the other.

- Distributed systems currently run on equipment managed by different administrations, which desire to keep its use under their control. Current practice is to rely on the users of this equipment to properly employ it according to policies promulgated by these administrations. However, this approach is changing as organizational IT budgets become larger and as Board of Directors, stock holders and other interested parties hold management more strictly accountable for information processing equipment use. Organizationally imposed constraints on distributed applications have a deleterious effect on their operation. A clear example of this is how router based firewalls not only provide protection against intruder attacks, but also prevent certain distributed applications, such as those using UDP, Java applets or those using the X window system from running across firewall boundaries. Massively distributed systems will experience even greater impediments to their operation, since not only will locally imposed constraints interfere with their operation, but other constraints will hinder them, such as national restrictions on transborder data flows and encrypted

data as well as the mandated use of certain standards for communications protocols.

- Distributed applications are composed of software components[2] that run on various platforms and are organized into component classes (e.g., file servers, public key distribution systems, file system clients, various clients on embedded systems), each of which performs a different function. Current distributed applications either assume that all components run the same version of software, or are structured according to a simple client/server model that must concurrently accommodate different versions on a small number of system (e.g., two or three). Massively distributed applications, because of other heterogeneity requirements, such as the use of different presentation formats, costing and cost recovery schemes, naming techniques and administration constraints, will be constructed of a significantly larger number of component types. Components of these types in general will not run the same version of software. Designing and implementing these applications will therefore require engineering techniques that allow them to operate in the face of software versioning heterogeneity.

- One of the fundamental mistakes made when building distributed systems is ignoring legacy applications and infrastructure. This invariably leads to poor customer acceptance of new technology. Customers cannot simply discard their existing applications when new infrastructure becomes available. In many cases these applications and their supporting infrastructure represent billions of investment dollars and reimplementing them according to the interfaces presented by new technology is economically infeasible. Massively distributed systems will be no different in this regard. However, existing distributed systems generally must accommodate stand-alone legacy systems and applications. Massively distributed systems, on the other hand, must accommodate both stand-alone as well as current distributed system legacy infrastructure and applications.

- While most existing distributed applications run on a number of different computing platforms, they are generally limited to a small number of common families, e.g., Unix, Windows, and perhaps MVS. Massively distributed applications, on the other hand, will run not only on the legacy platforms, but also on a wide variety of embedded systems supported by their own proprietary operating systems and hardware (e.g., automobile control systems, PDAs). For example, a massively distributed application for remote conferencing may have components that run on workstations, on equipment provided by a manufacturer for the TV cable industry, on cellular phones, on PCS based personal communication devices, and so forth. This will increase the number of different implementations of the software for a single component type and thereby increase the effort necessary to ensure the application works correctly.

[2] The term "component" is used here in its general sense. Distributed system components may or may not be constructed as object-oriented software components.

3.3. Objects (representation, encoding and translation)

A variety of efforts are underway to determine the best programming model for distributed objects (e.g., CORBA, Java). Lessons from these investigations will directly affect how objects are handled in massively distributed systems. However, there are certain issues arising from the scale of these systems that will introduce new constraints on how objects are represented, encoded and translated.

- Massively distributed system by their nature will require the creation, movement, modification and destruction of massive objects, which conceivably could contain thousands to millions of other objects. The size of these objects will affect how they are represented. For example, it will be infeasible to incorporate a copy of each contained object within the massive object. Instead, object references will be used within massive objects to represent its constituents. These objects will themselves be constructed using references to their contained objects, leading to a hierarchical object anatomy. However, several well-known problems associated with referencing distributed objects will require attention. For example, implementers of massively distributed systems will have to solve the lost object problem, which occurs when an object is destroyed without destroying all its references in other objects. Relying on manual techniques to recover from this error condition will not be an option, because of its scale. Similarly, revoking access to an object will be much more problematic for massively distributed systems than for existing distributed systems. It will be infeasible to locate all references to an object and delete them, even if the underlying object representation allows that. While using access lists can mitigate the problem of revocation, their use in massively distributed systems will be problematic,

since their size may make this approach unattractive. This problem is discussed in more detail in section 3.5.

- Not only will the representation of massively distributed objects require new techniques, their presentation to users will also require innovation. Some researchers have examined this problem. A new class of user interface represents objects as virtual spaces. This technique is eminently suitable for presenting massively distributed objects to end users. For example, a first level object might be represented as a virtual world, its constituent objects as countries, then cities, streets, houses, rooms, and so on, the exact structure depending on the size of the first level object and its relationship to its constituents as well as the relationship of the constituents to each other. Such presentation paradigms will be required to make massively distributed objects accessible to the technologically naive user.

- Since massively distributed applications will be global in scope, their users will belong to multiple cultures and countries. This will increase the necessity for the internationalization of data. While current applications can be configured to use different character sets, depending on choices made when the system is installed, massively distributed systems will have to internationalize data on-the-fly. This is especially true of embedded systems, which commonly will exist in mobile equipment. There will be no point when a system administrator can select a particular internationalized presentation set. Users of massively distributed applications will come and go during its lifetime, requiring run-time configuration of this data. Perhaps more importantly, there will not be a single system administrator who controls a massively distributed application. Its control will be distributed as well. Not only will character set data require internationalization, the application will have to internationalize other data such as financial, length/mass/time, and timezone. For example, an application that presents financial data may have to convert between dollars, yen and marks, depending on where the end-point system is located and how it has been configured by the customer.

- The storage of massively distributed objects will require new techniques. Current object storage systems assume objects are located on resources controlled by a single administration. The storage of a single massively distributed object, however, may use the resources of many storage systems, controlled by different administrations. This will lead to new problems in object store performance, error control and correctness. Object implementers will have to deal with the internal synchronization of object constituents, in regards to their creation, movement, modification and destruction. They also will have to deal with these issues when designing and implementing object caches.

- The algorithms used to manipulate massively distributed objects will utilize techniques that differ from those commonly used today. They will be replaced by those that use multi-cast communications in order to avoid object components understanding the object's global architecture. Voting and distributed agreement algorithms will become common. Programmers will create active objects, i.e., those which include active on-going computing, that blur the distinction between data and processing. While some existing distributed systems support active objects, massively distributed systems will organize large parallel computations around this concept, forming these computations by creating a first level object containing a large number of active objects residing on geographically disperse and administratively heterogeneous systems.

3.4. Resource management

In order to design and build massively distributed applications, engineers will have to grapple with new problems in resource management. Many existing distributed systems operate according to a local resource control model. Local processing manages its own resources, interacting with other threads of control through message passing or remote procedure call/method invocation. In massively distributed systems, objects will be composed of resources located in a large number of different places. Controlling the resources associated with an object will only be possible through a distributed global object resource management mechanism. This will introduce several new issues in distributed system resource control:

- Routing will become an issue not only at the network layer of the distributed system, but also at the application layer. Composite objects containing active objects will require routing services so the composites can interact with each other, especially if objects are allowed to move. For example, an embedded system such as a mobile PDA will move and connect to different portals in a network. If upon connecting, objects are moved from the PDA to execution environments hosted near the portal, invocation of those object's methods by other objects

will require a routing protocol to identify a path between them. The interactions between active objects and passive objects will require routing so active objects can locate and contact passive object methods. Congestion control within composite objects will be an issue. Programmers and object implementers will organize objects to avoid computational hot spots, similar to those experienced by massively parallel algorithms. Due to their scale, object implementers will require adaptive routing and congestion control within massively distributed objects, manual configuration will not be an option.

- Resource management in a massively distributed system will interact strongly with its heterogeneous nature. Applications will compose resources under the control of many administrations into a single distributed resource. Management of the distributed resource must accommodate the usage policies of the separate administrations. For example, cost recovery for a massively distributed object must accrete and disseminate sufficient funds to cover the costs of the composite objects. Since massively distributed objects may be highly recursive, cost recovery will require the composition of recursively discovered composite costs. As objects evolve, their cost recovery anatomy will change, forcing object implementers to dynamically configure the object's cost recovery algorithm. Since it will be difficult and expensive to continually track the exact cost structure of an object at any point in time, cost recovery will use statistical algorithms, assessing charges and dispersing payments so that the owners of component resources receive their payments, but avoiding too fine an accounting of usage. Periodic audits will ensure systematic fraud is minimized. Operators of massively distributed objects will adopt a fixed cost model, allowing users of the object to flexibly utilize its resources without charging for its components use.

- Object implementers will devise algorithms enforcing restrictions on object use that are required by restrictions on their components. Unlike existing distributed resource management, these algorithms will have to be adaptive, since implementers will not know all possible component resource usage policies, *a priori*. Object implementers will be challenged to understand object behavior, since dynamically changing an object's components, which may introduce new resource usage restrictions, will make this difficult.

- Massively distributed systems will contain extremely large volumes of data and enormous processing power. Effectively managing these resources will challenge implementers. Distributed system architects will merge the two prevalent ways to organize computations in a distributed system, i.e., *move data to the processing* (used by NFS, World Wide Web, FTP, gopher) and *move processing to the data* (used by active networking and Java applets) into a single scheme. Such objects will be moved within the distributed system and carry with them both code and data [6]. If the object consists primarily of data, it will closely approximate moving data to the processing. If it consists primarily of code, it will closely approximate moving processing to the data. RPC and other procedure-oriented paradigms will be emulated by including a single high level instruction in the code section, specifically a procedure identifier, which will be interpreted at the RPC server.

- The deployment of massively distributed systems will encourage the trend towards a new valuation model for information. Specifically, the independent variable driving value will be how long the information has existed. Massively distributed systems will increase the difficulty in keeping information private. Consequently, customers will pay for fresh information, which will rapidly decrease in value once it is produced. For example, stock market ticker data is now available electronically, making automated trading programs possible. However, this implies that most of the value of the ticker data is consumed in a very short period of time. Its public release without cost routinely occurs today 15 minutes after its generation. As massively distributed systems become common, more and more stock traders will be forced to use automatic trading programs, which will reduce the value of the ticker data even faster. It is likely that eventually ticker data will be available without cost within a few minutes of its production. This paradigm will apply to other data, such as library searches, financial analyses and market research.

- Upgrading software in a massively distributed system will pose new problems for software companies. It will become more difficult to locate all users of a particular version of software and users will become less cognizant of the version they are using. This situation will arise because objects will contain and use other objects implemented by software not directly visible to the user. To meet this new challenge, software companies will move from a pure

product business model to a combination of product and service model. Indeed, this is already happening. Many software companies offers a subscription purchase agreement for their products that provides the customer with one year of upgrades. Engineers will design their software to periodically query maintenance centers that will upgrade it automatically, if the software's maintenance contract is still active. Again, this is current practice. For example, the virus detection software that is part of the Norton System Tools product for Windows 95/98 periodically communicates with a maintenance site on the Internet and downloads new virus checking features as they are released. Customers will be forced to use maintenance contracts to keep their applications running. In the future, customers will force software manufacturers to provide open interfaces to their software and negotiate maintenance agreements with secondary companies, much like the hardware business does today.

- Massively distributed systems will force a reexamination of the way communication interfaces work. In particular, they will have to scale to accommodate a large number of correspondents. For example, a simple mass-market purchase application that allows customers to choose between several products may receive millions of requests in a short period of time. Current programming interfaces to communication services are totally inadequate to handle such a high rate of transactions. Furthermore, individually acknowledging each of these transactions would impose a significant processing burden on the systems running the application. This will encourage engineers to design communication interfaces and protocols that are better suited to high rate and volume transactions.

- Since massively distributed systems will dramatically increase the amount of traffic handled by a communications fabric, engineers will investigate how to efficiently move extremely large volumes of traffic over the internet. In addition much of this traffic will belong to one of several different quality of service classes (e.g., best effort delivery data, stream data) and so understanding how to accommodate different quality of service parameters in the internet is a problem with high priority. Fortunately, this problem is currently receiving significant attention by the Internet Engineering Task Force and the IEEE.

- Massively distributed systems will support a volume of information flow to an end-system beyond that which a user is capable of manually examining for interest, usefulness, or suitability. Various filtering techniques will be developed and used by customers to ensure their systems only present to them information in which they are interested. Additionally, once these filtration systems are common, applications will use feedback controls to gather and employ end-system supplied filtration data to decrease the amount of information sent to an end-point that is automatically discarded. Massively distributed systems will provide tools to applications that allow them to support this type of customization for large numbers of end-point systems.

3.5. Protection (security and privacy)

Protection of distributed system assets, including base resources such as processing, storage, communications and user-interface I/O as well as higher-level composites of these resources, e.g., processes, files, messages, display windows and more complex objects is not a strength of existing distributed systems. While engineers are currently engaged in developing solutions to the many problems that exist in this area, they are not addressing protection issues that will arise as massively distributed systems become prominent. Specifically:

- Massively distributed systems for the most part will support a large number of end-systems, many of which will be embedded in other equipment and used by technologically naive customers. These systems will require management, which very probably will occur through the use of systems under the control of service providers. Since many end-systems will either generate or contain data that customers consider private, the management technology for massively distributed systems must guarantee, to a reasonable extent, that unaggregated data is not compromised either by individuals operating the management sub-system or by the service providers as part of their corporate strategy, e.g., during the collection of marketing data. To meet this objective, end-systems must be customer anonymous with respect to the management sub-system. That is, there must be no tie between the identifiers used to manage end-systems and those used for billing, warranty or other services that require the identification of the individuals who own or use these end-systems.

- Since one-to-many communications will play a major role in massively distributed systems, engineers will have to address the problems of deletion, modification, insertion, replay, release of secret state, and

masquerade for this type of communications. Existing techniques are designed to protect one-to-one communications against these threats. New protocols and processing algorithms will be required to provide these services for one-to-many communications.

- The scale of massively distributed systems will introduce new problems in resource access control for both end-systems and infrastructure support systems. Currently, most systems use access control lists or their approximations (e.g., Unix file access permission bits). However, massively distributed systems will support millions of principals, which would lead to access control lists of unmanageable size, if implemented as they are currently. Implementers will require new techniques, such as access list caching hierarchies, capability systems that solve the revocation and lost object problems, and access control techniques that combine the advantages and characteristics of access control lists with capability access control and eliminate the disadvantages of each. Since massively distributed systems will be highly heterogeneous and require the support of legacy distributed systems, engineers will be forced to grapple with the hazards of composing systems that use access control lists with those using capability based access control [7].

- When confronted with the problem of information protection, existing distributed systems already must cope with the controls governments have placed on cryptographic technology. These constraints have hampered engineers in providing appropriate levels of security to the users of distributed systems. Massively distributed systems will generate new problems in this area. Since communications will occur between large numbers of end-systems, security service implementations will be hard pressed to ascertain which systems are constrained by particular national laws concerning encryption and what those limitations might be. As mobile systems become more prevalent and as they are integrated into massively distributed systems, enforcement of these constraints, even if they are known, would require tracking the position of a mobile system, identifying the constraints imposed by its locality, and communicating these constraints to all systems interacting with it. For the number of end-systems that will engage in a massively distributed application and for the frequency with which a mobile system's position might change (e.g., those used from or embedded in an automobile, train, boat or airplane), this will be technically infeasible. Since enforcing these legal constraints will be just as hard as implementing the technology that satisfies them, over time the constraints may be relaxed or even eliminated. However, while they are in force, the technical problems they raise will be formidable.

- Commercial enterprises will play a large role in massively distributed applications. Consequently, new threats will arise as the value of these applications increases. One concern is the security of internetwork routing. Massively distributed systems will introduce new factors into this problem. The infrastructure required for such systems will necessarily federate other large networks into a global communications fabric. However, routing service providers may wish to limit the traffic that transits their networks and subscribers may demand guarantees that the quality of service advertised by routing service providers is actually delivered. The former problem has received a great deal of attention, leading to the formulation of policy routing techniques. The latter problem on the other hand, has received less attention and will require further study.

3.6. Naming

Engineers have devoted considerable attention to the issue of naming in distributed systems over the last 20 years. Pioneering efforts, such as Grapevine [8], the Domain Name System (DNS) [9] and NIS [10], developed the concepts for the next generation of distributed naming systems, such as X.500 and LDAP accessed directories. To some extent these second-generation systems, especially X.500, are designed for large scale distributed systems. In particular, X.500 supports the storage and retrieval of customer defined data structures, a hierarchically structured and distributed naming context mechanism and decentralized authentication services. The Open Group's Distributed Computing Environment (DCE) uses either X.500 or DNS to construct a large scale naming system from its local Cell Directory System [11].

Even though the designers of second generation distributed naming systems have attempted to accommodate massively distributed systems in their design, there are a number of naming issues that transcend these designs and require attention:

- The volume of data within a massively distributed naming system will be so large, tens to hundreds of millions of entries, that customers will not know where to start when looking for an item. Hierarchical structures have the advantage of logarithmically

scaling the name space, given a specific choice of how the data should be organized. They have the disadvantage of constraining efficient queries to those whose unknowns are at the bottom of the hierarchy. Since a single massively distributed system will be used for a wide variety of purposes, multiple indexes into its naming data will be required. Independent services will manage these indexes and each will need to update their meta-data as the foundation data changes. This already exists for indexes of webpage data. However, as anyone who has used internet search engines realizes, the volume of data returned for many queries is too large for efficient use. Higher-level indexing services will use the services of lower-level services, leading to a hierarchy of naming data accessible by customers. This web will contain data of varying degrees of freshness, which will require new methods and algorithms to present a consistent and coherent view to customers.

- As stated previously, a large number of the organizational structures within a massively distributed system will be ephemeral, e.g., information communities which will unite around a particular event and then dissolve. However, many who would be candidates to participate in such an event may not be aware, *a priori*, that it exists. The scale of a massively distributed system invalidates traditional advertising techniques, since the volume and rate of the advertising information arriving at an end-system using these approaches would overload it. Consequently, engineers will devise new producer/consumer models of advertising. For example, one possibility is a hierarchically structured reverse advertising technique, whereby the customer advertises desirable product attributes to first tier brokers, which accrete these into reverse advertisements to higher level brokers. From other leaves in the hierarchy, product producers advertise their products to other first-tier brokers, which likewise accrete them into advertisements to higher level brokers. Somewhere within the hierarchy a rendezvous occurs and direct advertisements are sent or otherwise made available to customers. A simple example of this strategy for advertising and using computing resources has already been prototyped [12]

- Moving objects requires moving names embedded in the object. This necessitates either retaining the context in which those names are interpreted or translating them into contexts available at the new object site. Massively distributed systems makes either strategy difficult. Retaining contexts will compel an object to maintain contact with them as it moves around, which will in turn require meta-naming services (i.e., to name the naming context). Translating naming contexts will necessitate the discovery of the relationships between the different contexts. Furthermore, object sharing requires the naming of objects by different entities. Communicating names for the purpose of sharing requires moving the associated naming contexts.

- Different cultures may require different naming schemes, some hierarchical, some local. These must be coalesced into a usable massively distributed system naming scheme. The same object may be named differently depending on the use of the name and its characteristics (e.g., native language, available character set).

3.7 Error Control

To a certain extent the problem of error control in global networks has not been solved for existing distributed systems. For example, multicast links fail during video teleconferences, ftp sites become unreachable during a transfer due to firewall, router or link failure, and end-system failure results in broadcast storms that cripple local communications. All of these events are difficult to analyze and correct automatically. They invariably require the use of out-of-band channels to notify a responsible technician or system administrator. In a massively distributed system, however, the identification and use of an appropriate out-of-band channel will be problematic. Furthermore, service providers must provide error control in ways that conform to other requirements, such as privacy protection, high performance and scalability. Such constraints lead to the following problems:

- Fault isolation and correction in massively distributed systems will require new infrastructure services to monitor communications quality and deliver exception alarms to service providers when quality falls below a given threshold. Since the number of distributed applications running in a massively distributed system will be too large for manual monitoring to be cost effective, service providers and distributed system implementers will design, implement and deploy automatic fault detection and isolation mechanisms to ensure service remains available during various resource outages. This will be especially important for distributed embedded systems. The functions of existing network operation centers will become automated. Network operation centers will focus on higher level fault isolation and control,

such as rerouting communication services when a catastrophic event takes down large subsystems.

- Engineers of massively distributed systems will introduce new ways to process a large volume of information and number of transactions. As previously described for costing strategies, statistical algorithms will be used to cope with the decreasing probability that all necessary fault identification information is available in a reasonable amount of time. For example, consider the problem of automating the collection of payments for a video service. Errors in transmission, bugs in software, unresponsiveness of end-system equipment due to malfunction and operator error will lead to a certain amount of imprecision in the accounting and collection processes. An automated billing and collection application will record all funds received, match them with outstanding balances on accounts and attempt to resolve the amount received with the overall outstanding balance. A discrepancy below a certain threshold will be charged to the cost of doing business. Auditing processes will attempt to ensure that these discrepancies are not systematically occurring as the result of fraud. Such techniques will also be used for automated opinion polling, physical resource flow monitoring (e.g., power, water), and distributed system infrastructure maintenance.

- Massively distributed systems will require the use of highly available subsystems, including a redundant communications fabric provided by multiple service providers, fault-tolerant distributed processing for control and accounting, and high availability end-systems for content providers, which will be the next logical step beyond the high availability file systems currently deployed. Fault-isolation will have to operate in the face of privacy services, such as encryption, that are virtually non-existent today, but that will become prominent in the next decade. Service providers will have to respond to subscriber problem reports, even when the ultimate source and destination of traffic causing the problem is unknown. They will gravitate toward a scaled cost structure for various grades of service, from high grade indemnified service ("you lose service, we lose revenue"), to lower grades of best effort service.

- Detection of inconsistent state will be difficult in massively distributed systems, since the distributed state will not be available for centralized processing. Consequently, algorithms for communications and processing will be developed that are tolerant of inconsistencies. They will use redundancy not only within communication protocols, but also within objects so their state driven algorithms are fault tolerant. To achieve this objective, there will be an increased use of voting protocols and algorithms.

- It will be impossible to keep track of all the identifiers or other processing state associated with an object. Massively distributed systems must be designed to work in the face of lost objects, which will be the rule rather than the exception. Loss of identifiers will also increase the incidence of orphaned objects, which will increasingly require the use of global object garbage collection.

- One-to-many communications on a large scale will encourage the development of forward error correction techniques, which will replace feedback error control in many cases. Feedback error control requires the retention of a large amount of state and a high level of processing when applied to one-to-many communications. Idempotent operations will become the paradigm of choice when one-to-many communications support remote procedure or object method calls.

- Error control will move more and more into the application layer as the error characteristics of lower layers become heterogeneous (especially for one-to-many communications) and as applications are widely distributed. [13]. To manage application error control, management systems will expand their concerns from the network layer and below to all layers of the protocol hierarchy. This will lead to the standardization of management protocols and interfaces.

3.8 Synchronization

One of the most difficult problems that engineers of massively distributed systems will encounter is synchronizing computations consisting of thousands to millions of components. Current methods of synchronization, such as semaphores, monitors, barriers, remote procedure call, object method invocation, and message passing, do not scale well. Generally, they are suitable either for synchronizing closely coupled computations (e.g., semaphores, monitors and barriers), or for unicast distributed applications (e.g., remote procedure calls, object method invocations and message passing). Engineers have not yet devised efficient synchronization techniques for group computations, when groups contain thousands to millions of active elements. Specific problems in the area of synchronization include:

- Caching will become pervasive to accommodate performance, error control and resource management objectives. Keeping all cached copies synchronized will require new cache coherency algorithms, since simple write-back or write-through schemes do not scale properly. This will lead to synchronization marker algorithms where an authoritative copy of information is broadcast periodically to resync cached state. Resolution of differences will occur as a result of this synchronization activity. However, there will be many locations where portions of a large database are authoritative and redundancy in the data will accommodate failures.

- The scale of massively distributed applications will encourage engineers to use asynchronous computing, mimicking the way live organisms function. This will be a revolution in the way computations are designed, implemented and operated. It will no longer be possible to assume or reason that the state of some distributed application equals a given value at any point in time. Instead, engineers will design distributed application algorithms so that an invariant is always true with high probability.

- The use of pre-agreement algorithms based on roughly synchronized clocks will emerge as a major synchronization technique for massively distributed applications [14]. The accuracy and drift of local clocks will become a major engineering problem. Separate synchronization systems (e.g., the NIST time standard broadcast over WWV) will be used to decouple clock synchronization from resource depletion problems and other errors of massively distributed applications. The use of geographically distant physical clocks on high-precision real-time applications will require the transmission of both clock and location in a synchronization schedule to accommodate relativistic effects.

- It will be virtually impossible for thousands of components to synchronize using communications when it is not possible to use an external synchronizing signal. If synchronized clocks are not achievable, for example, because some components do not have real-time clocks (e.g., simple embedded systems), barrier synchronization implemented as a hierarchical tree will be necessary. Because of the attendant performance degradation associated with barrier synchronization, it will only be employed at major synchronization points separated by long intervals of time.

- To accommodate the improbability of a component knowing all other components with which it should synchronize, multicast communications will become a major synchronization technique. Thus, a component might multicast a message that announces the occurrence of an event to a multicast group without knowing exactly which other components belong to that group. Use of this technique will rely on the designer making a trade-off between synchronization and resource consumption.

- Synchronization and fault tolerance will interact strongly. Synchronization mechanisms will be required to operate in the face of a certain level of component and communications failure. However, synchronization algorithms will be designed to move the distributed application towards a well-defined goal even when failures occur. Thus, massively distributed applications will make significant use of probabilistic algorithms that complete with a probability less than unity in a given length of time, but that ensure this probability always monotonically increases in time.

3.9 Measurement, testing and debugging

The implementation of massively distributed applications requires measurement, testing and debugging to ensure applications behave correctly and perform well. These services are also required to correct implementations when they fail to meet either of these criteria. The scale of massively distributed systems makes the execution of these tasks difficult. In particular:

- Gathering the necessary measurement data to determine whether massively distributed applications are performing properly will be problematic. Directing this data to a single destination will almost certainly interfere with the phenomena being measured. Consequently, measurement will occur through an auxiliary distributed application, running in tandem with the target application. The measurement application will itself be massively distributed, requiring its own control infrastructure as well as new monitoring techniques that allow it to maintain contact with the measured application even when parts of it are created, destroyed and moved.

- Testing massively distributed applications will require new techniques to ensure the application scales. Current approaches used by testing engineers, which concentrate on determining whether the application works properly on a single machine will be inade-

quate for massively distributed applications While there is current work to develop harnesses to test client/server based applications, the introduction of a third support process into the test harnessthat supports client/server interactions (e.g., a database containing public key certificates) is rare. Due to the expense of developing a dedicated test harness for massively distributed applications, engineers will be forced to test applications during large scale beta deployment. This is in fact becoming the norm for much mass market software.

- It may not be possible to debug a massively distributed application under full load, so engineers will be forced to make trade-offs between debugging and real-time fault isolation with correction. Traditional debugging techniques, such as breakpointing, do not scale. Therefore, trace-driven analytical techniques will predominate when debugging massively distributed applications. Since test harnesses of the requisite massive scale will be economically infeasible, engineers will be forced to use execution traces of live application runs. Such execution traces may produce a very large volume of data, so test engineers will have to develop agile filtering, coalescing and viewing tools.

4. Summary

The nature of this paper is exploratory and most assertions are presented without detailed justification. Technology forecasting is always risky and, in hindsight, rarely completely accurate. However, I hope the analysis at least stimulated the reader to think about massively distributed systems, how their characteristics will influence embedded system design and implementation, and how they may change the way we do computing in the future. Even if there is disagreement about its content, it has succeeded if it motivates readers to develop an alternative taxonomy of the design issues and challenges associated with massively distributed systems.

5. References

[1] Gustavsson, R., "Agents with power," Communications of the ACM, Vol 42, No. 3, March, 1999, pp. 41-47.

[2] Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., and Minden. G.L, "A survey of active network research," IEEE Communications Magazine, pages 80-86, January 1997.

[3] Watson, R. W., "Distributed system architecture model," in Distributed Systems, Architecture and Implementation, Springer-Verlag, Berlin, NY, 1981.

[4] Takada, H. and Sakamura, K., "Compact, low-cost, but real-time distributed computing for computer augmented environments," Proc. 5th IEEE Comp. Soc. Workshop on Future Trends of Dist. Comp. Sys., Cheju Island, Korea, Aug., 1995, pp. 56-63.

[5] Lange, L., "The Internet," IEEE Spectrum, January, 1999, pp. 35-40. (see also www.oxygen.org).

[6] Sadok, D. H., Kelner, J., and Silva, R.A., "A distributed programming platform using mobile agents," 3rd Inter. Symp. On Autonomous Decentralized Sys., April, 1997, pp. 103-110.

[7] Nessett, D.M., "Factors affecting distributed system security," IEEE Trans. Soft. Eng., vol. SE- 13, pp. 223-248, 1987.

[8] Birrell, A.D., Levin, R., Needham, R.M.., and Schroeder, M.D., "Grapevine: an exercise in distributed computing," Communications of the ACM, April, 1982, 260-274.

[9] Mockapetris, P.V., Dunlap, K.J., "Development of the domain name system," Proc. of SIGCOMM '88, ACM, August 16-19, 1988, pp.123-133.

[10] Weiss, P., "Yellow pages protocol specification," Technical Report, Sun Microsystems, Inc., Mountain View, CA., 1985.

[11] Dilley, J., "Practical experiences with the OSF cell directory service," Networked Systems Architecture, Hewlett-Packard, International Workshop OSF DCE, Karlsruhe, Germany, Oct., 1993

[12] Cappello, P., Christiansen, B., Neary, M., and Schauser, K., "Market-based massively parallel internet compuing," Proc. 3rd Working Conference on Massively Parallel Programming Models, London, England, Nov., 1997, IEEE Computer Society, pp. 118-129.

[13] Saltzer, J., Reed, D., and Clark, D., "End-to-end arguments in system design," ACM Transactions on Computer Systems 2(4), Nov., 1984, pp. 277-288.

[14] Kopetz, H., "The time-triggered architecture," Proc. 1st IEEE International Symposium on Object-oriented Real-time Distributed Computing, Kyoto, Japan, April, 1998, pp. 22-29.

Learning in Intelligent Embedded Systems

Daniel D. Lee
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974
email: ddlee@lucent.com

H. Sebastian Seung
Dept. of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02138
email: seung@mit.edu

Abstract

Information processing capabilities of embedded systems presently lack the robustness and rich complexity found in biological systems. Endowing artificial systems with the ability to adapt to changing conditions requires algorithms that can rapidly learn from examples. We demonstrate the application of one such learning algorithm on an inexpensive robot constructed to perform simple sensorimotor tasks. The robot learns to track a particular object by discovering the salient visual and auditory cues unique to that object. The system uses a convolutional neural network to combine color, luminance, motion, and auditory information. The weights of the networks are adjusted using feedback from a teacher to reflect the reliability of the various input channels in the surrounding environment. We also discuss how unsupervised learning can discover features in data without external interaction. An unsupervised algorithm based upon nonnegative matrix factorization is able to automatically learn the different parts of objects. Such a parts-based representation of data is crucial for robust object recognition.

1 Introduction

The information processing capabilities embedded in systems today are extremely unreliable when operated in conditions that fall outside of their narrow design specifications. For example, the best present-day speech recognition systems will fail if a different microphone is substituted, or if the speaker has a sore throat [Rabiner]. On the other hand, biological systems are extremely robust to environmental changes when compared with artificial systems. The success of biological information processing systems lies in their ability to accommodate changes at all processing levels, from low-level sensor modalities to high-level computational algorithms and architectures. In order for artificial systems to truly become ubiquitous in the future, they will need to incorporate this ability to quickly and robustly adapt to changes.

In these proceedings, we present some of our work on algorithms that allow a system to learn from prior experience and adapt its behavior accordingly. We demonstrate these algorithms on a small quadruped robot that we have constructed to perform various kinds of sensorimotor tasks. In particular, we show how the robot learns to track a novel object by rapidly changing the weights of a convolutional neural network which processes the color, luminance, motion, and audio signals. Based upon the reliability of the different input channels, the system is able to discover the most salient visual and auditory cues unique to that object.

The training of the robot is done online in real time using supervisory signals from a teacher. We also explore how the robot can learn robust cues for object recognition without any supervision. This form of unsupervised learning is essential for adaptation in situations where there is little user interaction. We show how a simple algorithm can automatically learn to segment high-dimensional input data into features that correspond to functionally relevant parts [Palmer]. Our learning al-

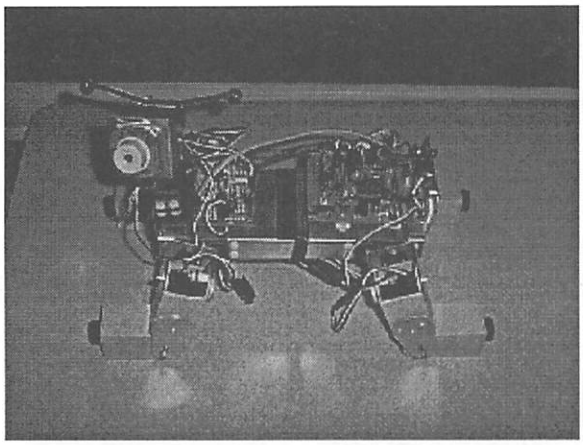

Figure 1: Photograph showing a small quadruped mobile robot that we have constructed to demonstrate our learning algorithms on sensorimotor tasks.

gorithm incorporates nonnegativity constraints that are similar to those found in biological neural networks. This enables our algorithm to learn parts as features by modeling positive coactivation in the inputs. Such a parts-based representation is valuable because it is invariant to perturbations or occlusions that affect localized regions of the input space.

2 Artificial sensorimotor system

In order to demonstrate how learning algorithms can be applied to improve the information processing capabilities of systems in real environments, we have constructed a small quadruped robot that is able to perform various simple sensorimotor tasks. Figure 1 shows a picture of our robot, which is approximately 25 cm in length and about 1 kg in weight, powered by a 9.6V rechargeable battery pack. It contains 14 hobby servo motors to provide three degrees of freedom for each of the four legs, as well as two degrees of freedom for the head. Attached to the head assembly is a small board level CCD camera that acts as a single eye, as well as two directional electret microphones for ears. The two head servo motors allow the camera to rapidly pan and tilt over a wide range of viewing angles. In addition to the visual and auditory inputs, a two-axis gyroscopic rate sensor is used to provide vestibular input for stabilization tasks.

An onboard Motorola 68HC11 microprocessor converts and processes the gyroscopic inputs, and provides auditory feedback by producing sounds in a small speaker. It also generates the timing signals for the 14 motors,

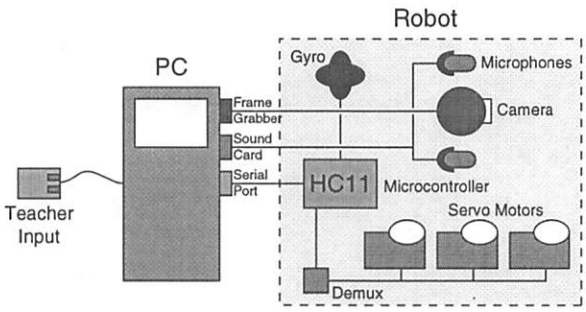

Figure 2: Schematic diagram showing the various hardware components of the system.

which are demultiplexed to the separate motors using shift registers. Because the microprocessor is not powerful enough to process the input signals from the CCD camera and microphones, the video and audio signals are currently sent to an offboard computer for processing as shown in Figure 2. The computer is then able to direct the onboard microprocessor to generate the appropriate motor behavior via a serial link.

The neural network algorithms for visual and auditory processing and control are implemented using Matlab for rapid development purposes. Real-time acquisition of the video and audio signals into Matlab variables is accomplished with custom-written drivers running under either the Microsoft Windows or the Linux operating system. Because the hardware was built using cheap, commercially available components (total cost of the system $< \$700$), the control software needs to be robust against the large sources of noise and drift in the hardware components. The following sections explain how online learning algorithms can compensate for limitations in the hardware and changes in environmental conditions.

3 Tracking application

Let us consider the problem of trying to program the robot to watch and follow with its head as someone walks around the room. This type of active perception is critical for proper functioning of the human visual system [Yarbus]. Because human retinas have small, high resolution foveal regions surrounded by visual fields of relatively much lower resolution, the eyes need to be making constant movements in order to keep objects focused on the foveas. Thus, the role of the human oculomotor system is to direct relevant objects in the visual world onto these high resolution areas. To accomplish

this task, biology has evolved many complex neural circuits to control eye movements. Some examples include the saccadic system which rapidly acquires new objects, and the smooth pursuit system which tracks slowly moving objects [Horiuchi, Rao].

In analogy with the biological system, our robot also needs to first determine which feature in its visual field is most relevant and then direct its gaze towards that object. As the target person moves about, the robot's tracking system will attempt to stabilize the person's image in the center of the field of view. Unfortunately, the physical performance of our hardware system is very lacking compared to biology. A fixed lens is used on the camera which gives it roughly a 65° horizontal and 48° vertical field of view. With the video digitized at frames of 120×160 pixels, this corresponds to a little less than half a degree of angular resolution, which is 25 times worse than human foveal acuity. Also, the maximum saccadic velocity of the servo motors is about 300 deg/sec, which is twice as slow as human eye movements. The tracking algorithm needs to be able to overcome these physical limitations.

A naive heuristic algorithm for the robot to track someone is to simply have it follow the largest moving object in the image. However, such a system is easily fooled if there are multiple objects moving in the room. More sophisticated algorithms [Darrell, Petajan] have proposed locating human heads by employing rules such as flesh color detection [Yang], or matching ellipses to head outlines [Eleftheriadis]. Another approach is to use the directional microphones to locate any person who is speaking using audio cues [Bregman]. But all these algorithms will fail in situations for which they were not designed. Such predetermined tracking algorithms tend to be very brittle and break when the surrounding environment is highly variable.

Recently, a potentially more robust method has been demonstrated for head detection using neural networks to learn the appropriate grayscale features of a face [Sung, Nowlan, Rowley]. Although these networks can very accurately detect faces in images, they need to be trained on a very large set of labelled face and non-face images. Since these networks are typically trained in batch mode on a preset ensemble of faces, they also do not learn to discriminate one person from another.

For our robot, we use a convolutional neural network that rapidly learns to locate and track a particular person's head. The system learns to do this task in real time with online supervisory signals. The network architecture integrates multimodal information in a natural way,

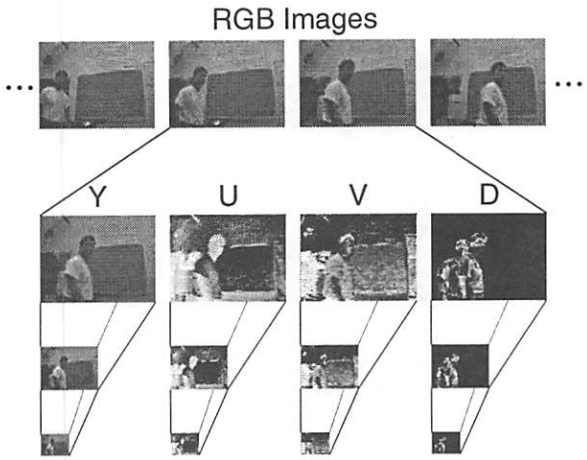

Figure 3: Preprocessing of the video images. Luminance, chromatic and motion information are separately represented in the Y, U, V, D channels at multiple resolutions.

and the fast online adaptation of the network's weights allows it to adjust the relative importance of the inputs depending upon the current environment. This enables the system to robustly track a person even in a cluttered background in the presence of other moving objects.

3.1 Video preprocessing

The raw video signal from the robot first needs to be preprocessed before it can be input to the convolutional neural network. The composite video signal from the CCD camera is digitized with a video capture board into a time series of raw 120×160 RGB images as shown in Figure 3. Each RGB color image is then converted into its YUV representation, and a difference (D) image is also computed from the absolute value of the difference between consecutive frames. The Y component represents the luminance or grayscale information in the image, while the U and V channels contain the chromatic or color data. Motion information in the video stream is represented in the D image where moving objects appear highlighted.

At each time step, the four YUVD images are then subsampled successively to yield representations at lower and lower resolutions. The resulting "image pyramids" allow the network to achieve recognition invariance across many different image scales without having to train separate neural networks for each resolution. Instead, a single neural network with the same set of weights is simultaneously run across the different resolutions, and the maximally active resolution and position

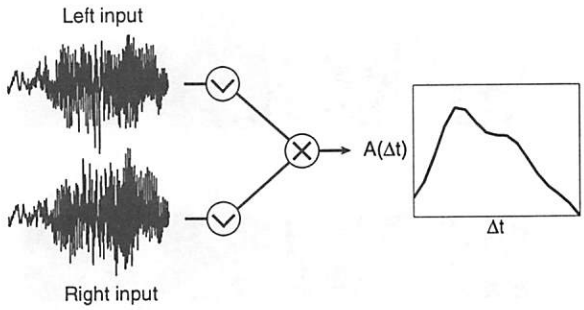

Figure 4: Processing of the audio waveforms to yield a topological map of audio sources as a function of time delay.

is selected.

3.2 Audio preprocessing

Animals are able to robustly localize audio sources using a variety of auditory cues [Bregman]. Experiments have shown that one of the most important cues for azimuthal position determination is interaural time difference, corresponding to the slight delay it takes for a sound to reach the different ears [Konishi]. Similarly, the two microphones on the head of our robot are separated horizontally in space to optimize the interaural difference between their audio signals. This information is then combined with visual cues by the convolutional neural network to determine the overall saliency of the different locations in its field of view. In order for the neural network to process the auditory information in the same manner as it does visual information, the raw audio data has to first be converted into an auditory space map as depicted in Figure 4.

The audio signal from the left and right microphones are first digitized at 16000 Hz in the sound card. These waveforms are then passed through a rectification nonlinearity to emphasize the envelope of the sound waveforms present in the recordings. The resulting signals are then cross-correlated, giving the following time correlation function:

$$A(\Delta t) = \sum_t |x_L(t)||x_R(t+\Delta t)| \qquad (1)$$

where $x_L(t)$ and $x_R(t)$ are the audio inputs from the left and right microphones. Ambient background noises are attenuated by subtracting out the mean of this correlation function. Different values of the time delay Δt correspond to varying azimuthal positions of auditory sources, assuming their elevation angle is near the horizontal plane of the microphones. Thus, the cross-correlation indicates the likelihood that an auditory source is located at the various azimuthal positions. This correlation function can be considered a one-dimensional auditory space map of the environment.

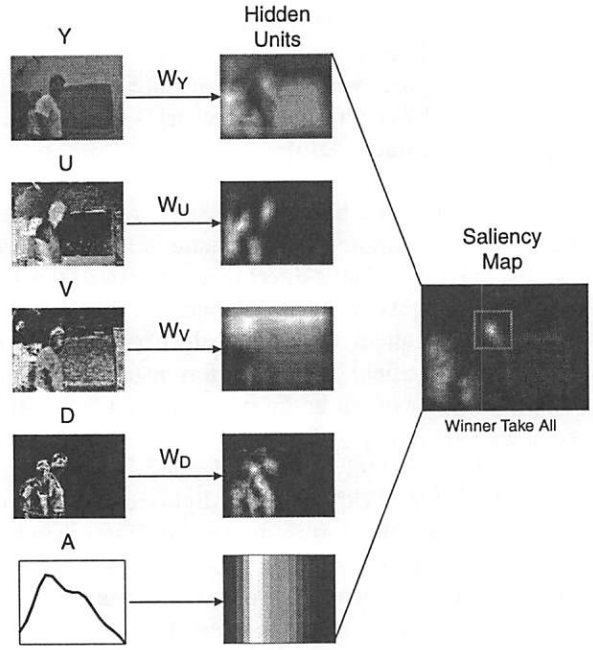

Figure 5: Neural network uses a convolutional architecture to integrate the different sources of information and determine the maximally salient object.

4 Supervised learning

4.1 Neural Network Architecture

Our tracking algorithm uses the convolutional neural network architecture shown in Figure 5 to locate the salient objects in its visual and auditory fields. The YUVD input images are filtered with separate 16×16 kernels, denoted by W_Y, W_U, W_V, and W_D respectively. This results in the filtered images $\bar{Y}^s$, $\bar{U}^s$, $\bar{V}^s$, $\bar{D}^s$:

$$\begin{aligned} \bar{Z}^s(i,j) &= W_Z \circ Z^s \\ &= \sum_{i',j'} W_Z(i',j')\, Z^s(i+i', j+j') \end{aligned} \qquad (2)$$

where s denotes the scale resolution of the inputs, and Z represents any one of the Y, U, V, or D channels. These filtered images correspond to a single layer of hidden units in the neural network. The hidden units are then combined with the one-dimenional auditory correlation function $A(j)$ to form the saliency map X^s in the following manner:

$$\begin{aligned} X^s(i,j) &= c_Y\, g[\bar{Y}^s(i,j)] + c_U\, g[\bar{U}^s(i,j)] + \\ &\quad c_V\, g[\bar{V}^s(i,j)] + c_D\, g[\bar{D}^s(i,j)] + \\ &\quad c_A\, g[A(j)] + c_0 \end{aligned} \tag{3}$$

where the sigmoidal nonlinearity is given by $g(x) = \tanh(x)$. Thus, the saliency X^s is computed on a pixel-by-pixel basis using a nonlinear combination of hidden units. The relative importance of the different luminance, chromatic, motion, and auditory channels in the overall saliency of an object is given by the scalar variables c_Y, c_U, c_V, c_D, and c_A.

With the bias term c_0, the function $g[X^s(i,j)]$ may be interpreted as the relative probability that the tracked object appears in location (i,j) at input resolution s. The final output of the neural network is then determined in a competitive manner by finding the location (i_m, j_m) and scale s_m of the best possible match:

$$g[X_m] = g[X^{s_m}(i_m, j_m)] = \max_{i,j,s} g[X^s(i,j)]. \tag{4}$$

After processing the visual and auditory inputs in this manner, head movements are generated in order to keep the maximally salient object located near the center of the field of view.

4.2 Adaptation

Our robot learns to track a specific target using supervisory feedback from a teacher. During training, the teacher watches the robot's video input and corrects its behavior whenever it makes a mistake. The teacher corrects the robot by indicating where the target is located in the visual field using a graphical user interface. This action prompts the learning algorithm to adjust the weights of the convolutional neural network in order to better track the object at that location.

The algorithm uses the supervisory signals to adjust the kernels W_Z and scalar weights c_Z of the neural network. The neural network is updated whenever the maximally salient location of the neural network (i_m, j_m) does not correspond to the desired object's true position (i_n, j_n) as identified by the teacher. Objective functions proportional to the sum squared error terms at the maximal location and at the new desired location are used for training the network:

$$e_m^2 = |g_m - g[X^{s_m}(i_m, j_m)|^2, \tag{5}$$

$$e_n^2 = \min_s |g_n - g[X^s(i_n, j_n)|^2. \tag{6}$$

For each correction that is provided by the teacher, the algorithm tries to minimize the errors in these objective functions. First, the gradients of Eqs. 5–6 are computed. These gradient terms are then backpropagated through the convolutional network [Nowlan, LeCun], resulting in the following rules for adaptation:

$$\begin{aligned} \Delta c_Z &= \eta\, e_m g'(X_m) g[\bar{Z}(i_m, j_m)] + \\ &\quad \eta\, e_n g'(X_n) g[\bar{Z}(i_n, j_n)], \end{aligned} \tag{7}$$

$$\begin{aligned} \Delta W_Z &= \eta\, e_m g'(X_m) g'(\bar{Z}_m) c_Z Z_m + \\ &\quad \eta\, e_n g'(X_n) g'(\bar{Z}_n) c_Z Z_n. \end{aligned} \tag{8}$$

In typical applications of neural networks where a large set of the training examples can be considered simultaneously, the learning rate η is set to some small positive number. However in our case, it is desirable for the robot to learn to track an object in a new environment as quickly as possible. Thus, rapid adaptation of the weights during even a single training example is needed. A natural way of doing this is to use a fairly large learning rate, and to repeatedly apply the update rules in Eqs. 7–8 until the calculated maximally salient location is very close to the actual desired position.

4.3 Training example

An example of how quickly the robot is able to adapt is given by the learning curve in Figure 6. In this particular example, the system was trained to track the head of one of the authors as he moved around and talked in his office. The weights were first initialized to small random values and the learning parameters were set to $g_m = 0$, $g_n = 1$, and $\eta = 0.1$. The robot was then corrected in an online fashion using supervisory inputs to follow the author's head.

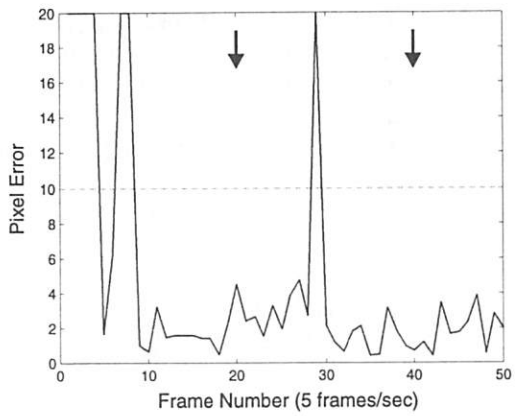

Figure 6: Fast online adaptation of the neural network. The head location error in pixels in a 120 × 160 image is plotted as a function of frame number (5 frames/sec).

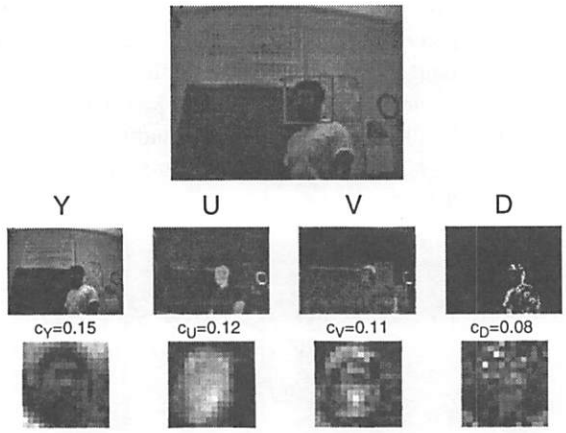

Figure 7: Example showing the inputs and weights used in tracking a head ($c_A = 0.05$). The head position as calculated by the neural network is marked with a box.

After only a few seconds of training at 5 frames/sec (200 ms processing time per image), the system was able to locate the head to within four pixels of accuracy, as determined by hand labelling the video data afterwards. As saccadic eye movements were initiated at the times indicated by the arrows in Figure 6, new environments of the office were sampled and an occasional large error was seen. However, over time these errors were corrected, and the neural network learned to robustly discriminate the head from the office surroundings.

Figure 7 shows the inputs and weights of the network after a minute of training as the author walked around his office. The kernels necessarily appear slightly smeared because they are adapted to be invariant to slight changes in head position, rotation, and scale. But they clearly depict the dark hair, facial features, and skin color of the head. The relative weighting ($c_Y, c_U, c_V > c_D, c_A$) of the different input channels shows that the luminance and color information are the most reliable for tracking the head. This is probably because the presence of other moving body parts and external noise sources in the office made the motion and auditory channels relatively unreliable.

More complicated neural network architectures could be used for combining the different sensory inputs to achieve better tracking performance. However, this example shows how a simple convolutional network architecture can efficiently integrate the different visual and auditory cues in order to learn how to robustly track an object. Moreover, by using fast online adaptation of the neural network weights, the system is able to rapidly accommodate changing environments.

5 Unsupervised learning

The online learning algorithm described above allows our robot to rapidly adapt and correct mistakes when given supervisory feedback by a teacher. But there are many situations in which it is impossible for any teacher to be present. In these situations, would it still be possible for a system to adapt based upon raw sensory stimuli without being told the appropriate thing to do? This is generically quite a difficult problem, but there are some well-established algorithms that allow the system to continue to adapt even without a teacher. Generally, these unsupervised learning algorithms attempt to extract common sets of features present in the input data. The system can then use these features to reconstruct structured patterns from corrupted input data. This invariance of the system to noise allows for more robust processing in performing recognition tasks.

One commonly used technique for building invariances into systems is to project the input data onto a few representative directions known as the principal components. This procedure called principal components analysis (PCA) has historically found widespread use in data compression, modeling, and classification systems [Jolliffe, Turk]. For an example of its application, consider the bottom portion of Figure 8. Here, a set of images of handwritten "two"'s has been analyzed using PCA. The images are taken from the Buffalo Zip Code database [LeCun], and can formally be described by a $n \times m$ matrix X of pixel values. Each of the $m = 731$ grayscale images is preprocessed to roughly center and align it within a 16×16 grid. The pixels are scaled such that white is equal to zero and black is equal to one. The

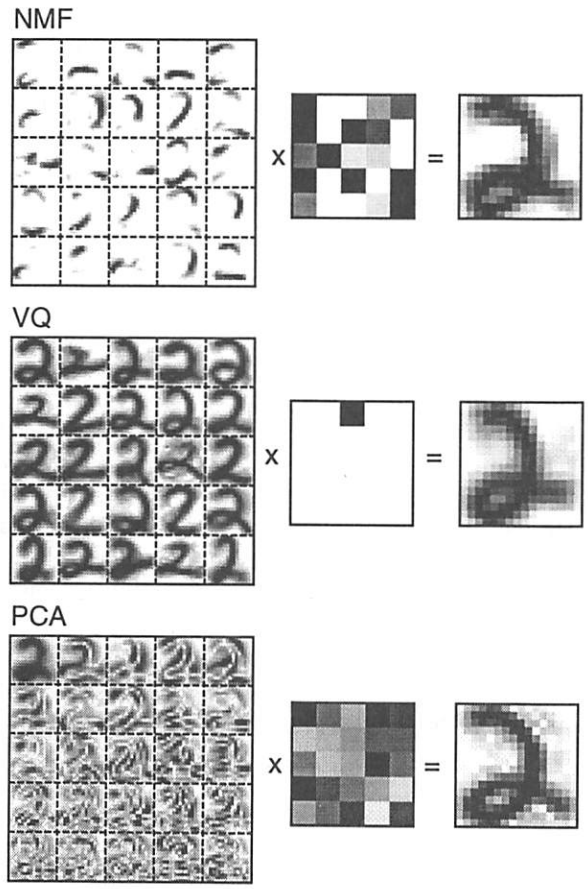

Figure 8: Three different unsupervised learning techniques applied to handwritten images of the digit "two." NMF learns a representation based upon the strokes of the digits, while VQ and PCA learn holistic representations. For each algorithm, the 25 columns of basis W are depicted as a 5×5 montage on the left, with their corresponding activations V in the middle. The reconstruction of the digit given by the product WV is displayed on the right of the figure.

$n = 256$ pixel values are then arranged into a column of X, with each handwritten image forming a separate column. PCA factorizes the matrix X into the approximate form $X \approx WV$ where the matrix factors W and V are $n \times r$ and $r \times m$ in size, respectively. The columns of W correspond to an orthogonal basis set which can be used to reconstruct the original images in X using the coefficients in V. The left side of Figure 8 displays the $r = 25$ different columns of W, which are then multiplied by a particular set of coefficients in V to form the reconstruction image on the right.

PCA is able to efficiently reconstruct the original data X using the limited number r of bases because it uses a representation that is distributed across all of the matrix factors. Some of the bases in W resemble common distortions of "two"'s such as translations and rotations, so the PCA representation is able to capture this global variability in the data. However, this representation utilizes both positive and negative combinations of all the available bases in W. This implies that the reconstruction involves finely tuned cancellations among the different bases, so it is difficult to visualize exactly how most of the columns in W contribute to a robust representation.

In contrast, vector quantization (VQ) is another unsupervised learning technique that categorizes the input data in terms of prototypes rather than orthogonal basis vectors. VQ may be thought of as applying a quantizing error correction to the data in order to remove small perturbations in the input. Formally, it can again be described as an approximate matrix factorization $X \approx WV$ where the columns in matrix V of activations are constrained to be unary (exactly one element in each column is equal to one, all the other elements are zero). In the middle portion of Figure 8, the effect of this constraint can be seen as forcing the VQ factorization to learn various templates of different types of "two"'s. In this case, reconstruction merely involves choosing the most similar prototype, and the representation is extremely sparse since only a single element of V is active.

The top portion of Figure 8 shows our new matrix factorization algorithm that decomposes the images into their representative parts. It achieves this by utilizing nonnegativity as a constraint on the components of the matrix factors W and V [Lee]. As seen in the figure, nonnegative matrix factorization (NMF) learns to decompose the image data into their constituent parts, corresponding to the different strokes of a "two." To approximate a particular two, the appropriate strokes are summed together to form the reconstruction. This is in sharp contrast to the holistic features found by PCA and VQ. Note that NMF finds a decomposition into localized parts even though no prior information about the topology of the images was built into the input data, i.e. the rows of matrix X could have been arbitrarily scrambled and the NMF algorithm would have yielded the same results.

NMF also combines some of the representational advantages of PCA and VQ. Since many of the parts are used to form the reconstruction, NMF has the combinatorial expressiveness of a distributed representation. But because the nonzero elements of W and V are all positive, NMF allows only additive combinations. So unlike PCA, no cancellations can occur. On the other hand, the nonnegativity constraint causes almost half of the coefficients in V to go to zero, resulting in a sparse coding of the input. Thus, NMF learns a parts-based repre-

$$W_{ij} \leftarrow W_{ij} \sum_k \frac{X_{ik}}{(WV)_{ik}} V_{jk}$$

$$W_{ij} \leftarrow \frac{W_{ij}}{\sum_k W_{kj}}$$

$$V_{ij} \leftarrow V_{ij} \sum_k \frac{X_{kj}}{(WV)_{kj}} W_{ki}$$

Figure 9: Multiplicative update rules for nonnegative matrix factorization. The accuracy of the current approximation $X \approx WV$ enters the learning through the quotient $X_{ik}/(WV)_{ik}$. It can be shown that these rules maximize the objective function $\sum_{i=1}^{n} \sum_{k=1}^{m} [X_{ik} \log(WV)_{ik} - (WV)_{ik}]$.

sentation of the data that is sparse as well as distributed [Foldiak, Olshausen].

Analytically, the NMF algorithm can be derived from a probabilistic generative model that incorporates Poisson noise [Hinton]. This model has previously been used in the deconvolution of astronomical images that have been blurred by the atmosphere and the telescope [Richardson, Lucy]. NMF may thus be considered a generalization of this technique to blind deconvolution. By maximizing the likelihood of this probabilistic model, the NMF learning rule for nonnegative matrix factorization is obtained [Dempster, Saul]: Given a data matrix X, the matrix factors W and V are first initialized to random positive values. The factors are then updated using the multiplicative update rules shown in Figure 9, which guarantees that the likelihood of the data X is increased. This procedure is then iterated until an optimal factorization $X \approx WV$ is learned.

6 Discussion

The reason NMF discovers strokes as the functional parts of digits is because the nonnegativity constraints allow it to learn from positive coactivation of image pixels in the data. The NMF algorithm adapts the basis W in order to learn the appropriate coactivations. In constrast, PCA has no constraints on the sign of the activation, and learns simultaneously from both positive and negative activations in the data. It should be noted that biological neural networks may use similar types of constraints to achieve analogous representations. The firing rates of neurons cannot be negative, and the strengths of synapses do not generally change sign. These one-sided constraints could possibly be important in developing the sparsely, distributed coding of sensory input that give rise to robust biological information processing.

The NMF algorithm is also generally applicable to problems outside the domain of image analysis. We have also applied it to the semantic analysis of text documents [Salton, Landauer], as well as to the analysis of routing patterns in data networks. In each of these cases, the algorithm learns to decompose the input data into their constituent parts. This enables any additional processing of the data to be robust against perturbations that can change only a small number of these features [Biederman]. Our current research in this area involves incorporating the NMF parts decomposition of the input data into a hierarchical representation that would be appropriate for high level control of systems such as our robot. Along with future improvements in sensory, motor, communications, and processing hardware, advances in these new learning algorithms will hopefully allow artificial embedded systems to someday exhibit the computational complexity and robustness found in biological systems.

7 Acknowledgments

We acknowledge the support of Bell Laboratories, Lucent Technologies. We also thank M. Fee, A. Jacquin, S. Levinson, E. Petajan, G. Pingali, and L. Saul for helpful discussions.

References

[Biederman] I. Biederman, *Recognition by components: a theory of human image understanding*, Psychological Review 94 (1987) p. 115.

[Bregman] A. Bregman, *Auditory scene analysis: the perceptual organization of sound*, MIT Press, Cambridge, MA (1990).

[Darrell] T. Darrell, P. Maes, B. Blumberg, and A. P. Pentland, *A novel environment for situated vision and behavior*, Proc. IEEE Workshop for Visual Behaviors (1994) p. 68–72.

[Dempster] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via*

the EM algorithm, J. Royal Statistical Society 39 (1977) p. 1-38.

[Eleftheriadis] A. Eleftheriadis and A. Jacquin, *Automatic face location detection and tracking for model-assisted coding of video teleconferencing sequences at low bit-rates* Signal Processing: Image Communication 7 (1995) p. 231.

[Foldiak] P. Foldiak and M. Young, *Sparse coding in the primate cortex*, The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA (1995) p. 895–898.

[Hinton] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, *The "wake-sleep" algorithm for unsupervised neural networks*, Science 268 (1995) p. 1158–1161.

[Horiuchi] T. K. Horiuchi, B. Bishofberger, and C. Koch, *An analog VLSI saccadic eye movement system*, Advances in Neural Information Processing Systems 6 (1994) p. 582–589.

[Jolliffe] I. T. Jolliffe, *Principal Component Analysis.* Springer-Verlag, New York, NY (1986).

[Konishi] M. Konishi, *Listening with two ears*, Scientific American, April (1993) p. 66–73.

[Landauer] T. K. Landauer and S. T. Dumais, *The latent semantic analysis theory of knowledge*, Psychological Review 104 (1997) p. 211–240.

[LeCun] Y. Le Cun, et al, *Backpropagation applied to handwritten zip code recognition*, Neural Computation 1 (1989) p. 541.

[Lee] D. D. Lee and H. S. Seung, *Unsupervised learning by convex and conic coding*, Advances in Neural Information Processing Systems 9 (1997) p. 515–521.

[Lucy] L. B. Lucy, *An iterative technique for the rectification of observed distributions*, Astronomical J. 74 (1974) 745.

[Nowlan] S. J. Nowlan, and J. C. Platt, *A convolutional neural network hand tracker*, Advances in Neural Information Processing Systems 7 (1995) p. 901–908.

[Olshausen] B. A. Olshausen and D. J. Field, *Emergence of simple-cell receptive field properties by learning a sparse code for natural images*, Nature 381 (1996) p. 607–609.

[Palmer] S. E. Palmer, *Hierarchical structure in perceptual representation*, Cognitive Psychology 9 (1977) p. 441–474.

[Petajan] E. Petajan and H. P. Graf, *Robust face feature analysis for automatic speechreading and character animation.* Proc. 2nd Int. Conf. Automatic Face and Gesture Recognition (1996) p. 357-362.

[Rabiner] L. R. Rabiner, *Fundamentals of speech recognition*, Prentice Hall, Englewood Cliffs, NJ (1993).

[Rao] R. P. N. Rao, G. J. Zelinsky, M. M. Hayhoe, and D. H. Ballard, *Modeling saccadic targeting in visual search*, Advances in Neural Information Processing Systems 8 (1996) p. 830–836.

[Richardson] W. H. Richardson, *Bayesian-based iterative method of image restoration*, J. Opt. Soc. Am. 62 (1972) 55–59.

[Rowley] H. A. Rowley, S. Baluja, and T. Kanade, *Human face detection in visual scenes*, Advances in Neural Information Processing Systems 8 (1996) 875–881.

[Salton] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval* McGraw-Hill, New York, NY (1983).

[Saul] L. Saul, and F. Pereira, *Aggregate and mixed-order Markov models for statistical language processing*, In C. Cardie and R. Weischedel (eds), Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (1997) p. 81–89.

[Sung] K. K. Sung and T. Poggio, *Example-based learning for view-based human face detection*, Proc. 23rd Image Understanding Workshop (1994) p. 843–850.

[Turk] M. Turk and A. Pentland, *Eigenfaces for recognition*, J. Cognitive Neuroscience 3 (1991) p. 71–86.

[Yang] J. Yang and A. Waibel, *A real-time face tracker*, Proc. 3rd IEEE Workshop on Application of Computer Vision (1996) p. 142–147.

[Yarbus] A. L. Yarbus, *Eye movements and vision* Plenum Press (1967).

Using Mobile Code Interfaces to Control Ubiquitous Embedded Systems

Kari Kangas and Juha Röning

University of Oulu, Department of Electrical Engineering
Computer Engineering Laboratory
P.O. BOX 4500, FIN-90401 Oulu, Finland
{macconen, jjr}@ee.oulu.fi

Abstract

Devices controlled by embedded computers are becoming an integral part of our everyday life, as processor and memory capacities continue to increase while their cost decreases. In some embedded systems, however, the limited input and output capacities are beginning to restrict the design of complex functionality. Furthermore, as wireless communication devices are becoming commonplace even in embedded systems, the communication and interoperation between different systems will be increasingly important in the future. This paper describes a flexible, yet powerful concept that explains how a mobile code can be conveniently utilized by mobile users to control ubiquitous and diverse embedded systems in different environments. Apart from providing flexibility, the concept also aims to keep the embedded systems as simple as possible. We will illustrate the concept by presenting as an example a virtual user interface for a videocassette recorder. We will also discuss the possible benefits and drawbacks of the system. The concept described here can be extended to allow the mobile code to be used as interconnecting "glue" in diverse embedded systems. This glue could connect systems from several manufacturers to create smart environments that can be controlled by a single simple device.

1. Introduction

This paper describes the use of a mobile code to create Virtual User Interfaces (VUIs) that can be used to conveniently control ubiquitous embedded systems. Diverse embedded systems are becoming an integral part of our everyday life. We already encounter a large number of embedded systems not only at home but also elsewhere in our daily life. In the near future, embedded systems will be used to make up smart environments; i.e. environments that contain a very large number of objects controlled by a hidden, or embedded, computer [1]. In such environments, it is desirable that the same underlying technology and a single simple device can be used to control several embedded systems. We do not want every system to require a separate remote control unit or to include a complete user interface, as is usually the case at the present. Furthermore, the same technology should also be available to a mobile user. The mobile user should be able to control the unknown embedded systems that he or she may encounter in various environments. In addition, the communication between the embedded systems and the controlling device should be as local as possible, so that low-power and high-bandwidth communication devices, such as Bluetooth [2], could be used. Internet connection should only be required in exceptional situations. Local communication is closely related to the principle of self-containment. The embedded system in itself should contain at least the most important components required for controlling it. In this paper, we describe a concept that is our first step towards a system that can be used to control truly ubiquitous embedded systems in future mobile computing.

The functionality of embedded systems has traditionally been restricted by the high cost of processor power and memory capacity [3]. Low cost is especially important in consumer electronics. As the cost of powerful processors, memory, and other microelectronics has dropped dramatically, a large variety of extended functionality is being designed for embedded systems [4]. However, input and output capabilities have not improved at the same pace as the other capacities and some cases where complex functionality is implemented even require design compromises.

Embedded systems, such as Videocassette Recorders (VCRs) and microwave ovens, usually have displays that provide only very limited functionality for displaying text and high-resolution graphics. These displays are usually constructed by using Light Emitting Diodes (LEDs) or Liquid Crystal Displays (LCDs). Furthermore, these systems usually contain limited input devices, such as miniature keyboards, that may be inconvenient to use. Cost is the main reason why high-quality interfaces cannot be included in all embedded systems. However, this is not necessarily the only reason. Display and keyboard size is strictly limited in devices such as cellular phones or personal heart rate monitors.

Thus, the input and output capabilities are becoming the limiting factor in embedded systems in the future. This is especially problematic in the field of consumer electronics where most devices now fulfil their basic functionality requirements, and extended features are included to attract consumers. Setting up a VCR to record a TV show at a desired time without first consulting the manuals is already too difficult for some users. As embedded devices are becoming increasingly ubiquitous in the future, there will be an even more diverse user population while at the same time the necessity of being able to control various embedded systems will increase. To make the new users comfortable with new systems, we will need descriptive error messages in plain text, context-sensitive online help facilities, and automated configurations instead of cryptic error codes and signals.

Creating a virtual user interface by using the capabilities of an external control device (also referred as a client), such as a Personal Digital Assistant (PDA), can provide abundant input and output capabilities for an embedded system. In addition, the same device can be used to create virtual user interfaces for various systems, and it therefore also provides a notably cost-efficient solution compared to the alternative of including these input and output capabilities in each individual embedded system.

This paper describes a technique for creating virtual user interfaces by using a mobile code. For the purpose of this paper, mobile code is a software component that comes from one computer system and is executed by another. Java applet is an example of such a mobile code component. For the mobile code to be utilized, the embedded system must contain the code that is requested and used by an external control device to create a virtual user interface. This virtual user interface is then used to control the embedded system. The virtual user interface and the embedded system communicate by using an internal protocol that is completely invisible to the outside system. Below, a PDA will be used as an example of a control device.

The mobile code technique has three main goals. The first and most important goal is to minimize the processor and memory requirements of an embedded system while still providing enough flexibility and scalability to allow the technique to be used in diverse applications. This is achieved as follows. The software in the embedded system can be constructed by using traditional programming conventions and languages. The embedded system is not required to contain a virtual machine or to use software written in any specific programming language. In addition, different system manufacturers only need to agree on a very simple mobile code protocol. The only purpose of this protocol is to help the PDA user to identify different systems and to transfer the mobile code. Furthermore, the technique presented in this paper does not place any restrictions on the actual mobile code that is used to create a virtual user interface, nor does it restrict the appearance of the virtual user interfaces.

The second goal is to make it possible to construct self-contained embedded systems that can be controlled by using only local communication between the embedded system and the PDA. This would make it possible to use only low-power and high-bandwidth devices for communication. Local communication is achieved by storing the mobile code in each individual embedded system. It should be noted that the flexibility mentioned above is not sacrificed, as some parts of the mobile code may come, for example, from a server in the Internet. Storing parts of the mobile code in an external server violates the self-containment principle, but may be beneficial in some situations.

The third goal is to enable mobile computing. In mobile computing, the user roams in different environments and may encounter embedded systems previously unknown to him or her. It is therefore important that the PDA can adapt to new environments and new systems. In addition, the communication between the embedded systems and the PDA may be sporadic and the connections highly volatile. Adaptation to new systems is achieved by downloading the mobile code from each embedded system. The PDA is not required to identify the embedded system, but only to be able to execute the mobile code. A connectionless message based communication protocol is used to match the volatile communication channels.

This paper has been organized as follows. Chapter 2 will present related work. Chapter 3 will describe in detail the concept of utilizing a mobile code to create virtual user interfaces. Chapter 4 will describe an example system constructed to demonstrate how the mobile code concept can be implemented. Chapter 5 will discuss the benefits and drawbacks related to the mobile code technique. Chapter 6 will draw a conclusion and illustrate future work.

2. Related Work

Creating remote displays by using the X Window protocol in Unix or in systems utilizing teleporting [5] can be considered similar to creating virtual user interfaces. To make a clear distinction between these systems and the virtual user interface concept presented in this paper, such systems are said to utilize a display protocol. The systems utilizing a display protocol are constructed so that an embedded system executes a program and a PDA displays a user interface of that program by using its display capabilities. These two systems share a

common display protocol that is used to exchange display information. The PDA uses the information sent by the embedded system to construct and modify the user interface. When the user performs an action using the remote user interface, the PDA sends data containing the user commands to the embedded system for processing.

As obvious, designing a general-purpose display protocol usable in every possible application is by no means a trivial task. For example, the protocol must be flexible enough to meet the requirements of the different applications. Flexibility may, in some situations, mean that the protocol will be very complex and difficult to implement and use. Flexibility is usually achieved by including a redundant code to be utilized in future applications, and this redundant code may require too much storage to be usable in some embedded systems. Furthermore, the protocol must only require a small amount of communication bandwidth to be usable in systems with very limited communication resources. Even if such a general-purpose protocol can be designed, it must be widely accepted by the competing industry and be standardized. This can be very difficult, especially in the highly dynamic field of consumer electronics.

One way to reduce the required communication bandwidth is to use a lightweight version of the display protocol to create a remote user interface. An example of such a protocol is the Virtual Network Computing (VNC) protocol [6]. These protocols usually differ from normal display protocols in that the level of message abstraction is raised in order to reduce the amount of data transmitted between the embedded system and the PDA. In other words, whereas normal display protocols usually describe the remote user interface in great detail, lightweight protocols only describe the overall structure and rely on the more complex display services provided by the remote computer. This naturally reduces the need for communication but also reduces flexibility.

Another drawback in designing a standard communication protocol is that the embedded system generates the protocol messages that are used by the PDA to display the user interface. This means that the command messages arriving from the PDA as a result of user interaction must be translated into corresponding user interface messages. These messages are then sent back to the PDA. Assuming that the embedded system is relatively simple, the resource requirements for keeping track of the remote user interface status and generating correct messages may increase to undesirable extend.

One way to reduce the bandwidth requirement of remote user interfaces is not to use a display protocol at all, but to use some other relatively lightweight and possibly ubiquitous and well-known communication protocol instead. One possibility is to use the Hypertext Transfer Protocol (HTTP), usually combined with Common Gateway Interface (CGI) programs [7, 8]. The PDA requests Hypertext Markup Language (HTML) pages from the embedded system and displays them using a web browser. CGI programs can be used to construct these pages dynamically, so that they reflect the current status of the system. In addition, CGI programs are used to receive the commands issued by the web browser user. The benefit in this approach is that ubiquitous web browsers can be used to control the embedded system. Furthermore, the utilization of HTML pages and CGI programs is a simple and well-known technique. The drawbacks include the requirements of the embedded system to execute a TCP/IP stack and at least a minimal HTTP server. In addition to this, HTML pages have several limitations when used to create remote user interfaces. First of all, HTML pages cannot display data with temporal characteristics by default i.e. an HTML page can only be used to display a static snapshot of the system status. One way to display the dynamics is to use the HTTP push technique, but it may, in some circumstances, take up too much communication bandwidth. As an example, consider a situation where a new HTML page is pushed to the PDA every second, or every tenth of a second.

Another and more flexible way to create remote user interfaces is to use the HTTP and CGI programs in combination with Java applets, as outlined in [9]. The PDA uses the HTTP to request a Java applet. When the Java applet is run, it communicates with the embedded system using an application-specific protocol. The use of HTTP and Java applets naturally requires the embedded system to contain a communication component in addition to an HTTP server. The communication component communicates with the Java applet using an application-specific protocol. HTTP can also be used for communication between the Java applet and the embedded system, using CGI programs as described earlier. In essence, the principle in the HTTP and Java applet combination is the same as in the mobile code approach presented in this paper: to use a commonly agreed protocol to obtain a mobile code from the embedded system and then to use this code to create a remote user interface. An application-specific protocol is then used for subsequent communication between the remote user interface and the embedded system. The main problem with HTTP is that it is usually used in combination with the TCP/IP protocol. Most of the embedded systems in smart environments can be expected to be relatively simple; thus the TCP/IP protocol may be too heavyweight for such systems if they do not otherwise require an Internet connection.

The commercial systems Inferno [10] and Jini [11] also

offer a solution for controlling networked embedded systems, and the control aspects of Jini are very similar to the HTML and Java applet approach described above. However, both Inferno and Jini rely heavily on running a virtual machine in the embedded system and use a restricted set of programming languages. Furthermore, as they have both been designed by default to operate by using an Internet connection and the TCP/IP protocol, they would require slight customization to be suitable for wireless mobile computing. The reader should note that both Inferno and Jini provide a rich set of features other than control interfaces, which means that placing a virtual machine in an embedded system may be suitable in some applications. However, we feel that the manufacturer of the embedded system should not be forced to use a specific virtual machine or programming language.

Systems utilizing a mobile code for controlling embedded systems have also been described in [12] and [13]. Both of these systems rely on mobile code servers and the Internet for obtaining the user interface code. We feel that the possibility to construct self-contained embedded systems is so important that it justifies the hardware overhead in our approach compared to these systems.

3. Virtual User Interface as a Concept

A system utilizing mobile code virtual user interfaces is presented schematically in Figure 1. The main requirement for the complete system is that the embedded system and the client (e.g. a PDA) are connected via a communication channel of arbitrary form. This channel can be implemented by using an Infrared (IR) or wireless radio link or a cable. The channel is used for communication between the embedded system and the client. The communication includes the creation of a virtual user interface and the subsequent communication between the virtual user interface code and the embedded system. The client requests the mobile code from the embedded system by using a mobile code protocol. Both the embedded system and the client agree on the format of this protocol. The mobile code protocol provides only very limited services. These services can be used to identify the basic parameters of the selected system and to transfer the mobile code. Messages transferred through the communication channel that do not match the mobile code protocol can be considered to follow an internal communication protocol. This internal protocol is used between the virtual user interface and the control module in the embedded system, and it can be relayed directly through the mobile code protocol.

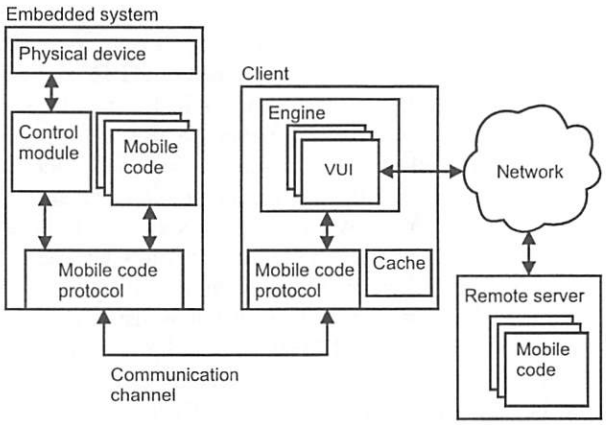

Figure 1. An outline of a system utilizing mobile code to create virtual user interfaces.

In addition to the component providing the mobile code protocol, the embedded system consists of a mobile code and a control module. The mobile code is stored on a non-volatile storage medium, such as an Electrically Erasable Programmable Read Only Memory (EEPROM) or Flash ROM, and sent to the client when required. The client then executes the code to create a virtual user interface. The embedded system may contain different types of mobile code to be sent to different types of clients. The amount of mobile code that can be stored in any particular embedded system depends on how much non-volatile storage can be afforded or feasibly included.

The control module acts as a communication relay between the virtual user interface in the client and the actual physical embedded system. The control module typically receives command messages, such as user requests to start VCR playback, originating from the virtual user interface. It transforms these commands into internal signals and sends them to the physical device. The control module also receives status signals from the physical device. The status signals reflect the effects of the commands issued and other external events, such as error conditions. The control module transforms these signals into status messages and sends them to the virtual user interface in the client. These command and status messages form the private communication protocol between the control module and the virtual user interface. The protocol can be chosen to match the exact requirements for each particular application. In other words, this internal communication protocol is likely to vary from one application to another. However, the system designer could also use a standardized communication protocol as an internal protocol; provided a suitable protocol is available.

The client consists of a component providing the mobile code protocol and a mobile code execution engine. The engine executes the mobile code and provides an

execution environment for it. The engine provides an abstraction layer of the actual client by providing basic services for the mobile code. As an example, such services may include a communication channel that can be used to communicate with the embedded system where the mobile code originated. Different engines may offer different services and provide varying levels of abstraction. As an example, even the client's operating system can operate as an execution engine for the mobile code. This requires that the mobile code is a complete executable binary image for that particular operating system.

It is also worth noting that the embedded system can contain only parts of the mobile code required to create a virtual user interface. In other words, some parts of the mobile code can be obtained from other code sources than the embedded system. Remote mobile code servers can operate as such code sources. Whether the mobile code is obtained fully from the embedded system or partly from the remote code server can be invisible to the client. This can be achieved by designing the mobile code in the embedded system to operate as a bootstrap code. This bootstrap code downloads the rest of the mobile code from the remote code server. This technique is suitable in situations where the total size of the mobile code is very large, or some parts of the mobile code are rarely required in everyday use. In an extreme situation, the embedded system may only contain a few lines of bootstrap code that is used by the mobile code engine to download the actual code from the remote server.

In order to minimize the need to constantly transfer the mobile code from the embedded system to the client for execution, mobile code caching can be utilized. For this purpose, the mobile code protocol may include a service that can be used to query the time when (if ever) the mobile code stored in the embedded system was last changed.

The technique described earlier in this Chapter can be considered to follow both the code-on-demand (COD) and the remote evaluation (REV) paradigms [14]. Both the embedded system and the client can initialize the creation of a virtual user interface. When the client initiates VUI creation, as is usually the case, the COD paradigm is followed. When the embedded system initiates VUI creation, the REV paradigm is followed. An example of VUI creation that is initialized by the embedded system is a situation where the embedded system uses a virtual user interface to notify the client of an exceptional situation. Such a situation could be the outcome of a scenario where the embedded system has detected a minor hardware failure during a self-check and wants to notify this to the first client that establishes a communication channel with it. To illustrate this with a concrete example, a smoke detector could notify a maintenance person walking by of a battery low condition.

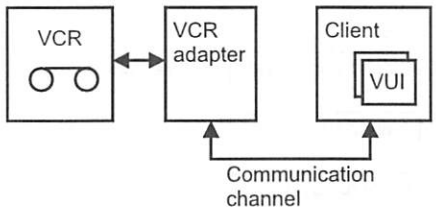

Figure 2. An example system providing a virtual user interface for a VCR device.

4. Implementation of a Virtual User Interface System

This Chapter explains how the concepts described in the previous Chapter can be utilized in an actual system. The example system provides a virtual user interface for a simple VCR device. This system was constructed to gain deeper understanding of the virtual user interface and the mobile code as a problem domain. The reader should note that the example system utilizes only a small fraction of the possibilities implicit in the mobile code approach. Furthermore, we agree that the VCR is not a good example in a sense that most of the current VCRs use also a TV screen to output status information.

All the software components in the system were constructed using the Java programming language. The main reason for this was the desire to be able to run the example system in a variety of hardware platforms. In a real system, the program run in the embedded system would probably be constructed by using a low level programming language, such as C or Assembler. The example system is presented schematically in Figure 2.

The system logically consists of four components: an actual VCR device, a VCR adapter, a communication channel, and a client system. It was difficult to find a VCR that could provide suitable feedback for the VCR adapter, mainly because the current VCRs usually provide only visual status signals. We can easily issue commands to the VCR via an infrared transmitter, but there is the risk that the VCR and the adapter enter an inconsistent state when, for example, the tape reaches the end. This was the reason why the actual VCR was not included in the system and a VCR program was constructed to simulate the physical device. Given that we can find a VCR with suitable feedback capabilities, it can be used to replace the VCR program in the future.

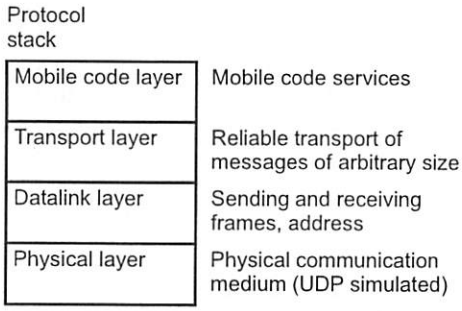

Figure 3. Communication protocol stack for the example system.

Communication channel

The VCR adapter and the VCR program were run in one computer and the client was run in another. In the example system, these computers were connected to the network by using the TCP/IP protocol. We constructed a simple communication protocol stack and used it for communication between the two computers. The stack is presented in Figure 3. The stack operates on top of a TCP/IP protocol stack and follows loosely the OSI reference model [15]. The stack consists of four layers: a physical layer, a datalink layer, a transport layer and a mobile code layer.

The physical layer was simulated using the UDP broadcast packets provided by the TCP/IP protocol. This was done to model the operation of communication devices that use radio waves or some other broadcast medium for communication. In broadcast communication, each data frame is received by every active system within the communication range.

The datalink layer provides services for sending and receiving directed and broadcast frames. Directed frames contain both source and destination addresses and are received only by the host indicated by the destination address. Broadcast frames contain only the source address and are received by all the active systems, except the broadcasting system.

The transport layer provides services for reliably transmitting and receiving messages of arbitrary length. It also provides a service for locating other active systems.

The mobile code layer provides the following services:

- Locating active systems
- Requesting information from the selected system
- Requesting a mobile code of the desired type from the selected system
- Transmitting the mobile code with the initialization parameters to the selected system for execution
- Transmitting data messages of arbitrary length to the selected system

The mobile code protocol is a very lightweight message-based protocol. The reason for this was the desire to keep the embedded system as simple as possible. Our goal was to provide only basic functionality that could be used to implement more complex communication mechanism, such as the Remote Procedure Call (RPC) system, if that were required in some applications.

Data messages of arbitrary length are sent through the mobile code layer using the notion of channels. When a data message is sent, an integer number indicating the channel is attached to it. The receiving mobile code layer can use this channel number to identify the destination component. In a sense, the channel number is similar to the port number in the TCP/IP protocol.

The protocol stack described above requires only a physical device with a capacity to broadcast fixed-length data frames to be utilized in various applications. Furthermore, the use of existing protocol stacks with the mobile code layer is very straightforward: it only requires services that can be used to locate and reliably transfer messages of an arbitrary length.

The main reason for constructing a complete ad-hoc protocol stack was the desire to model the operation of a low-range packet-based radio network as well as possible without actually using any wireless communication devices. We wanted to test the system first in a familiar environment, which could be easily monitored. We will use a standard protocol stack to replace the layers below the mobile code protocol if a suitable one can be found when, for example, Bluetooth is released.

Client

The purpose of the client system was to simulate a device that could be used to display virtual user interfaces

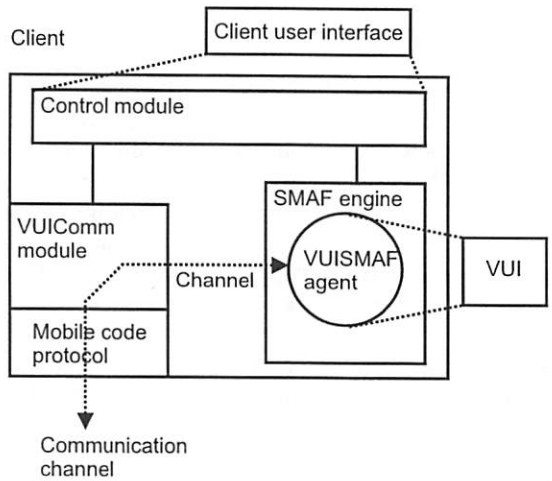

Figure 4. Structure of the client system.

for different embedded systems. Such devices include PDAs, future cellular phones and laptop computers. The structure of the client system is presented in Figure 4. The client system consists of the following software components:

- Mobile code protocol
- Control module
- Modified SMAF engine
- VUIComm module

The mobile code protocol operates as described earlier in this Chapter. The control module coordinates the cooperation of the other software modules. In addition, it also creates a user interface for the client. The user utilizes this user interface to control the client.

The Simple Mobile Agent Facility (SMAF) system (also referred as the SMAF engine) is used to run the VUI code. The SMAF system is a mobile code framework constructed by the authors to operate as a test bench for various mobile code paradigms. It also provides more profound knowledge of the implementation techniques needed for the mobile code systems. The SMAF system was not intended to compete in functionality with any of the existing mobile code frameworks.

The SMAF system was constructed using the SUN JDK (Java Development Kit) version 1.2. Java was selected as the implementation language for several reasons. First of all, Java contains several features that enable relatively easy implementation of various mobile code paradigms. Secondly, Java is platform independent in the sense that the compiled Java byte code is interpreted by the Java virtual machine (JVM). JVM is currently available for all the major computer platforms.

The overall architecture of the SMAF system follows the guidelines of the Mobile Agent System Interoperability Facility (MASIF) specification [16] from the Object Management Group (OMG). MASIF was formerly known as the Mobile Agent Facility (MAF) specification. The SMAF system operates as an execution environment for SMAF agents by providing basic services, such as agent transfer from one SMAF system to another.

When a SMAF agent is created, the Java byte codes that make up the agent are stored in an archive congruent with the Java Archive (JAR) file format. These byte codes are used for subsequent class loading during the agent's lifetime. The JAR file is transferred with the agent execution state when the agent moves from one SMAF system to another. In this way the agent does not need any external code servers for class loading during its lifetime. When implementing mobile code paradigms, the SMAF system utilizes a weakly mobile technology [14]. Systems utilizing a weakly mobile code technology do not preserve the precise execution state of the mobile code when the code is transferred from one system to another. Instead, only parts of the execution state are stored at the source host and transferred with the mobile code to the destination. The receiver uses this data to reconstruct some parts of the mobile code execution state. If the execution needs to be resumed, it can be implemented by, for example, using state variables to select the desired execution branch at the receiver.

The weakly mobile technology is implemented in the SMAF system by using Java Object Serialization. The mobile code state is serialized and the code is transferred to the destination with the serialized state data. When the mobile code is reconstructed at the receiver, the serialized state data is used only to restore the values of the objects' member variables. The state of local the variables in the member functions is lost.

The SMAF system used in the example system was modified slightly, so that it could be used to execute special virtual user interface agents (VUISMAF agent). The VUISMAF agent differs from the ordinary SMAF agent only in the sense that it contains additional functions that can be used to communicate with the embedded system where the agent code originated. The client in the example system can only execute a mobile code that is congruent with the VUISMAF or the SMAF agent structure.

The VUIComm module acts as a relay between the control module and the module that implements the mobile code protocol. It does this by creating and maintaining a channel list that is used to direct data received from the mobile code protocol to the correct virtual user interface. When a new VUI is created, the VUIComm module creates and assigns a channel object to that VUI. This channel object can be used to send and receive messages between the VCR adapter and the VUI. In other words, by using the channel objects, the virtual user interface can remain unaware of the actual mechanism that is used for communication between the client and the embedded system. The current version of the VUIComm module is implemented by assuming that VUISMAF agents will not move from one SMAF system to another, although this is possible.

VCR adapter

The purpose of the VCR adapter is to simulate a device that could be used to implement the virtual user interface functionality for different embedded systems. The VCR adapter acts as an interface between the actual VCR device and the rest of the system. The structure of the VCR adapter is presented in Figure 5.

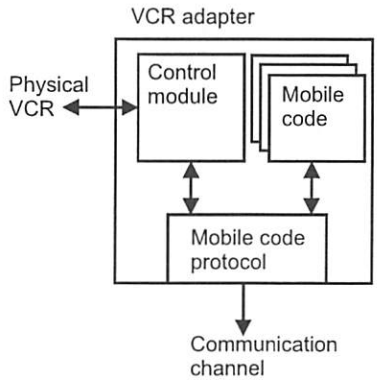

Figure 5. The structure of the VCR adapter.

As shown in the Figure, the VCR adapter consists of three components:

- Mobile code protocol
- Control module
- Mobile code

The mobile code protocol was described earlier in this Chapter.

The control module contains detailed information about the actual VCR device. It receives commands from the virtual user interface through the mobile code protocol and converts these commands into physical signals that manipulate the VCR. In addition, the control module transmits the messages that reflect the status of the VCR device to the virtual user interface. The mobile code contained in the VCR adapter follows the VUIS-MAF agent structure.

Virtual user interface

Figure 6 presents the virtual user interface for the example VCR device. It resembles a user interface for a standard VCR. The mobile code creates this VUI by using the standard Java class libraries located at the client. In other words, the mobile code uses the user interface primitives provided by the standard Java class library to construct a desired user interface.

When the user presses the control buttons, such as Play and Stop, the VUI sends command messages to the VCR adapter. These command messages have a very simple structure, consisting of only a few bytes. The VCR adapter receives these command messages and transforms them into internal messages. These internal messages are then sent to the program simulating the actual VCR device. The VCR adapter constructs status messages from the replies it receives from the VCR device and sends them to every virtual user interface that is currently active. In this way a given VCR can be inspected and manipulated by several virtual user interfaces at the same time.

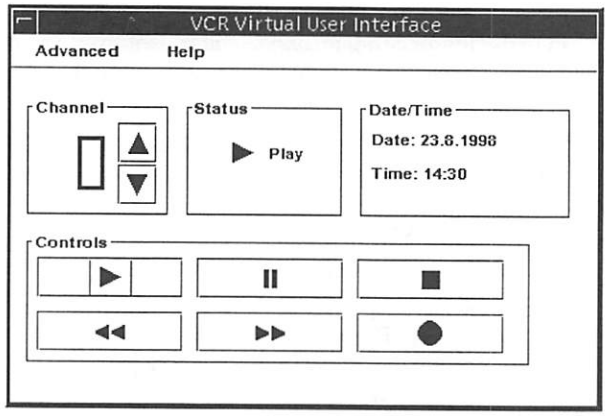

Figure 6. Virtual user interface for the VCR device.

The size of the JAR file containing the mobile code for the virtual user interface presented in Figure 7 is approximately 25 Kbytes. However, the current version of the interface was not designed or programmed so that the amount of code would be minimized. It is probable that the size of the JAR file can be reduced to fewer than 20 Kbytes.

5. Discussion

The technique for creating virtual user interfaces by using a mobile code offers several benefits compared to the existing approaches described in Chapter 2, but it also has some drawbacks. The benefits and drawbacks must be weighed carefully for every application if this technique is to be utilized in an actual system. This Chapter explains the benefits and drawbacks to allow proper evaluation of the suitability of this technique.

Benefits

The mobile code VUIs offer several benefits when compared to the existing approaches. The mobile code approach:

- is flexible, simple, and easy to implement
- is adaptable and utilizes efficient local communication
- is open
- allows a diverse set of value-added services

Flexibility is one of the main benefits of the mobile code VUIs. The mobile code uses the services provided by the execution engine to implement a desired user interface. Even though different execution engines and client systems may impose restrictions on the behavior of the mobile code, the approach presented in this paper does not in itself restrict functionality in any way.

Furthermore, flexibility does not make the approach complex or hard to implement. Traditional program-

ming conventions and languages can be used to program the software for the embedded system, and productive high-level programming languages, such as Java, can simultaneously be used for the VUI. The internal communication protocol between the embedded system and the VUI can also be tailored to suit the needs of each particular application.

Even an existing embedded system would require only minor and inexpensive modifications to include VUI functionality. The easiest way to include a VUI in an existing embedded system would be to add an I/O port for an external adapter, as in the VCR example. This port could be used to receive commands from the adapter and to send out status messages reflecting the status of the embedded system. The adapter would provide the VUI functionality, and the embedded system itself would need no further modifications. This adapter could be sold as an option for each embedded system.

Adaptation is one of the most important benefits when the mobile code approach is used in mobile computing. The mobile code VUIs allow mobile users to control diverse embedded systems without knowing anything specific about them. The only requirement is that the client and the embedded system can communicate with each other and that the client can execute the VUI code. It is a lot easier to agree on standard interfaces for execution engines than for a diverse set of embedded systems, at least if the execution engine provides sufficient services for the mobile code. Adaptation would be hard without a mobile code. The client would have to share a control protocol with every embedded system, or to have a separate control program installed for every possible system.

Efficient local communication is another important benefit for mobile computing. Embedded systems can be controlled by using a local communication, which means that low-power, high-bandwidth communication devices can be used and the client is not required to maintain a possibly expensive connection with the Internet or a remote server. However, flexibility ensures that the VUI can obtain some parts of the mobile code from the remote servers, possibly by using the network connection provided by the client. In this way, the infrequently used parts of the mobile code, or possibly the parts that implement functionality that requires network connection, can be obtained from the remote server.

The mobile code VUIs also offer an open solution, as the approach does not require any special programming language or hardware to be used. An embedded system may even contain a different mobile code to be used with different clients. Furthermore, if the mobile code in the embedded system is stored in a re-programmable ROM memory, services facilitating software updates can be easily implemented. The main reason for easy update is the clear distinction between the mobile code and the control software in the embedded system. The mobile code is not executed by the embedded system and dynamic software update techniques are therefore not required.

As an example, consider a situation where a new type of client with a new mobile code engine is introduced. If the user wants to use this client instead of the old one, he or she simply obtains a new VUI mobile code and updates it in the embedded system. One way to do this is to include a special update code in the embedded system that can be used to create a special VUI for software updates. This VUI code can then obtain the new mobile code from the remote server and upload it to the embedded system. Software updates can naturally be done if the mobile code is stored in a separate ROM chip that can be replaced with a new one. In this way, the software in the embedded system can be kept simpler, as it does not need to provide software update services.

The most interesting advantage in the mobile code approach is how different types of value-added services, which have previously been very hard to implement, if not totally unfeasible, could be provided for embedded systems. One example is the software update service described above. Another example is the integration of embedded systems in external services. For example, the VCR VUI can use an Internet connection provided by the client to contact a remote TV guide server in order to obtain TV program data. The user could then use this data to select the programs to be recorded and the VUI would convert the selections to record commands and sent them to the VCR. The mobile code for this service could be stored in a server in the Internet, so that it could be easily modified if there were a change, for example, in the TV guide server. Storing the mobile code in a remote server makes sense in this system, as this service requires an Internet connection for obtaining the TV guide data.

Drawbacks

The approach presented in this paper does not come without drawbacks. These drawbacks include:

- A need for additional hardware and maintenance
- Communication overhead in some situations
- Code mobility is still a relatively immature technology
- Information security

As described in Chapter 3, the main requirement for the mobile code approach is that the embedded system contains communication hardware. Furthermore, storing the mobile code in the embedded system requires a variable amount of non-volatile memory. The addition of this extra hardware can be problematic in systems

where the total cost of the product must be kept to the absolute minimum and the virtual user interface does not result in substantial competitive advantage. However, some form of communication hardware is required in any case if we want to add remote control functionality to an embedded system.

Maintenance may also be a problem when the mobile code is stored in the embedded system. For example, non-automated software updates are completely unsuitable if embedded systems become truly ubiquitous. We hope that most of the software updating can be automated, as outlined earlier in this Chapter.

Downloading the mobile code from the embedded system can also increase the communication overhead in situations where the actual communication session only lasts for a short period. This can be reduced or, in some situations, prevented by using mobile code caching at the client. For example, the frequently used mobile code, such as that used to control embedded systems at home, can remain stored in a client's cache.

The code mobility, while not new as a concept, is still relatively immature and not applied in a large scale. The designers and developers lack experience compared to their colleagues, who are using more traditional programming paradigms, such as the client-server. This can be very problematic, especially as it may be difficult to provide acceptable security in mobile computing.

Information security is probably the most important problem in mobile code VUIs that needs to be solved. Even though it is not recommended, we may ignore most of the security problems if the mobile code approach is used only at home or in other relatively closed environments, much like security is mostly ignored in the current remote control units. For example, we may be tempted to assume that we never encounter malicious mobile code or that no eavesdropping is possible.

However, information security is an essential requirement in mobile computing. In mobile computing, the user may want or need to control unknown systems, which makes it necessary to be able to shield clients against malicious mobile code. In addition, we cannot ignore eavesdropping if the communication involves sensitive data. Furthermore, embedded systems must also be shielded against unauthorized use.

The techniques providing information security for mobile computing, especially for systems utilizing mobile code, are still very immature. Most of the security techniques for mobile code systems have been designed for applications with specified features and restrictions, and are not necessarily suitable for other applications. In a sense, however, the security requirements for mobile code VUIs are very similar to the security requirements for Java applets, and we are therefore optimistic and expect that a suitable security mechanism can be constructed. Because security in mobile code systems in general is a very complex issue, only a brief outline of the problems and solutions is presented here. For a more complete description of the problem and the possible solutions, refer to [17]. The security requirements include:

- Protecting the client against malicious mobile code
- Authenticating the client (or the user)
- Authenticating the embedded system
- Preventing eavesdropping
- Ensuring mobile code integrity
- Protecting the executing mobile code against malicious clients

The client should be shielded in such a way that the downloaded mobile code is not able to perform malicious actions once executed. This can be implemented by a sandbox security model that is used to limit the operations that can be performed by the mobile code. An example of such sandbox security can be seen in [18].

The authentication of both communicating parties can be performed by using digital signatures. Authentication is important in systems that need to restrict the number of persons allowed to control the system. Furthermore, if the source of the mobile code can be authenticated, varying sandbox restrictions can be utilized on the mobile code, depending on the degree of trust the client has in that particular source.

Communication can be encrypted to prevent eavesdropping [19]. Eavesdropping must be prevented in systems that transmit sensitive information that should not be exposed to outsiders.

Digital signatures can also be used to ensure mobile code integrity. Mobile code integrity ensures that the code is not altered after its distribution.

Protecting the mobile code against malicious clients is a very hard problem, mainly because the client can inspect the code during its execution. Initial solutions on how to protect the mobile code from inspection suggest that the mobile code should be constructed so that the actual purpose of execution is not revealed. Code obfuscation [20] or cryptographic solutions [21] can be used to achieve this. Malicious clients may, for example, inspect mobile code to find ways to control the embedded system without authorization.

As mentioned earlier, information security in mobile code systems is a very complex issue and largely dependent of the different applications at hand. The level of security acceptable for a VCR device with a short-range IR transceiver may be totally unacceptable for some other system, such as the access control systems

of our future homes.

6. Conclusion and Future Work

This paper described a flexible, yet powerful concept of a mobile code used to control ubiquitous and diverse embedded systems. The approach to create mobile code VUIs offers several benefits compared to the alternative techniques. These benefits include simplicity, efficient local communication, adaptation, openness, and most notably, flexibility. Furthermore, the fact that a single simple device can be used to control different embedded systems offers a cost-efficient and convenient control solution. The approach is well suited for the highly dynamic computing environments of mobile computing in the future. However, it is also usable in static environments.

In a wider sense, our goal is to inspect the possibility to use mobile code as an interconnecting glue to integrate various embedded systems. The mobile code VUI concept presented in this paper is the first step towards this goal. We feel that the concept can be utilized and extended in future embedded systems to provide other functionality than just control. Such functionality could include delivering communication protocols and device drivers as mobile code to the other devices in the system. It is also interesting to envision the range of possible value-added services that could easily be implemented for different systems.

However, further work is needed to construct actual clients and embedded systems that utilize the mobile code concept in a way that allows it to be tested in a real-world situation. We have already constructed an early version of a system that uses radio modems for communication. In addition, we have constructed several early prototypes of virtual user interfaces for various embedded systems and also for other real-world objects. These prototype VUIs will be utilized in our future research as we try to fully understand the potential offered by the mobile code. We are also actively looking for suitable security techniques for VUIs with varying security requirements.

7. References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, September 1991, pp. 66-75.

[2] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen, and W. Allen, "Bluetooth: Visions, Goals, and Architecture," *Mobile Computing and Communications Review*, Vol. 2, No. 4, 1998, pp. 38-45.

[3] M. Schlett, "Trends in Embedded Microprocessor Design," *Computer*, August 1998, pp. 44-49.

[4] V. V. Badami and N. W. Chbat, "Home Appliances Get Smart," *IEEE Spectrum*, August 1998, pp. 36-43.

[5] K. R. Wood, T. Richardson, F. Bennet, A. Harter, and A. Hopper, "Global Teleporting with Java: Toward Ubiquitous Personalized Computing," *Computer*, February 1997, pp. 53-59.

[6] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, January-February 1998, pp. 33-38.

[7] I. D. Agranat, "Engineering Web Technologies For Embedded Applications," *IEEE Internet Computing*, May-June 1998, pp. 40-45.

[8] R. Itschner, C. Pommerell, and M. Rutishauser, "GLASS: Remote Monitoring of Embedded Systems In Power Engineering," *IEEE Internet Computing*, May-June 1998, pp. 46-52.

[9] A. Puliafito, O. Tomarchio, L. Vita, and K. S. Trivedi, "Increasing Application Accessibility Through Java," *IEEE Internet Computing*, July-August 1998, pp. 70-77.

[10] S. M. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. W. Tickey, and P. Winterbottom, "The Inferno Operating System," *Bell Labs Technical Journal*, Vol. 2, No 1, 1997, pp. 5-18.

[11] Sun Microsystems, Inc., "Jini Technology," 1998, http://java.sun.com/products/jini/

[12] T. D. Hodes, R. H. Katz, E. Servan-Schreiber, and L. Rowe, "Composable Ad-hoc Mobile Services for Universal Interaction," *Proc. of the 3rd annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM'97)*, September 26-30, 1997, Budapest, Hungary, pp. 1-12.

[13] G. Kortuem, Z. Segall, and M. Bauer, "Context-Aware, Adaptive Wearable Computers as Remote Interfaces to 'Intelligent' Environments," *Proc. of the 2nd Int. Symp. on Wearable Computers (ISWC'98)*, October 19-20, 1998, Pittsburgh, Pennsylvania, USA.

[14] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998, pp. 342-361.

[15] M. Rose, "The Open Book: A Practical Perspective on OSI," Prentice-Hall, Inc, New Jersey, 1990, pp. 651.

[16] Object Managegent Group (OMG), "Mobile Agents Revision," July 28, 1998, (MASIF Revision), Available online ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf

[17] D. Chess, "Security Issues in Mobile Code Systems," G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 1-14.

[18] G. Karjoth, D. Lange, and M. Oshima, "A Security Model for Aglets," *IEEE Internet Computing*, July-August 1997, pp. 68-77.

[19] C. Kaufman, R. Perlman, and M. Speciner, "Network Security: Private Communication in a Public World," Prentice Hall PTR, New Jersey, 1995. pp. 505.

[20] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Code Agents From Malicious Hosts," G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 92-113.

[21] T. Sander and C. Tschudin. "Protecting Mobile Agents Against Malicious Hosts" G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 44-60.

Challenges in Embedded Database System Administration

Margo I. Seltzer, Harvard University
Michael A. Olson, Sleepycat Software

Database configuration and maintenance have historically been complex tasks, often requiring expert knowledge of database design and application behavior. In an embedded environment, it is not possible to require such expertise or to perform ongoing database maintenance. This paper discusses the database administration challenges posed by embedded systems and describes how Sleepycat Software's Berkeley DB architecture addresses these challenges.

1 Introduction

Embedded systems provide a combination of opportunities and challenges in application and system configuration and management. As an embedded system is most often dedicated to a single application or small set of cooperating tasks, the operating conditions of the system are typically better understood than those of general-purpose computing environments. Similarly, as embedded systems are dedicated to a small set of tasks, one expects that the software to manage them would be small and simple. On the other hand, once an embedded system is deployed, it must continue to function without interruption and without administrator intervention.

Database administration has two components, initial configuration and ongoing maintenance. Initial configuration includes database design, manifestation, and tuning. The instantiation of the design includes decomposing the data into tables, relations, or objects and designating proper indices and their implementations (e.g., B-trees, hash tables, etc.). Tuning the design requires selecting a location for the log and data files, selecting appropriate database page sizes, specifying the size of in-memory caches, and determining the practical limits of multi-threading and concurrency. As we expect that an embedded system is created by experienced and knowledgeable engineers, requiring expertise during the initial system configuration process is acceptable, and we focus our efforts on the ongoing maintenance of the system. In this way, our emphasis differs from other projects that focus on automating database design, such as Microsoft's AutoAdmin project [3]and the "no-knobs" administration that is identified as an area of important future research by the Asilomar authors [1].

In this paper, we focus on what the authors of the Asilomar report call "gizmo" databases [1], databases that reside in devices such as smart cards, toasters, or telephones. The key characteristics of these databases are that their functionality must be completely transparent to users, no explicit database operations or database maintenance is ever performed, the database may crash at any time and must recover instantly, the device may undergo a hard reset at any time (requiring that the database return to its initial state), and the semantic integrity of the database must be maintained at all times. In Section 2, we provide more detail on the sorts of tasks typically performed by database administrators (DBAs) that must be automated in an embedded system.

The rest of this paper is structured as follows. In Section 2, we outline the requirements for embedded database support. In Section 3, we discuss how Berkeley DB is conducive to the hands-off management required in embedded systems. In Section 4, we discuss novel features that enhance Berkeley DB's suitability for the embedded applications. In Section 5, we discuss issues of footprint size. In Section 6, we discuss related work, and we conclude in Section 7.

2 Embedded Database Requirements

Historically, much of the commercial database industry has been driven by the requirements of high performance online transaction processing (OLTP), complex query processing, and the industry standard benchmarks that have emerged (e.g., TPC-C [9], TPC-D [10]) to allow for system comparisons. As embedded systems typically perform fairly simple queries and only rarely require high, sustained transaction rates, such metrics are not nearly as relevant for embedded database systems as are ease of maintenance, robustness, and a small memory and disk footprint. Because of continuously falling hardware prices, robustness and ease of maintenance are the key issues of these three requirements. Fortunately, robustness and ease of maintenance are side effects of simplicity and good design. These, in turn, lead to a small size, contributing to the third requirement of an embedded database system.

2.1 The User Perspective

Users must be able to trust the data stored in their devices and must not need to manually perform any database or system administration in order for their gizmo to perform correctly.

In the embedded database arena, it is the ongoing maintenance tasks that must be automated, not the initial system configuration. There are five tasks traditionally performed by DBAs, which must be performed automatically in embedded database systems. These tasks are log archival and reclamation, backup, data compaction/reorganization, automatic and rapid recovery, and reinitialization from an initial state.

Log archival and backup are tightly coupled. Database backups are part of any recoverable database installation, and log archival is analogous to incremental backup. It is not clear what the implications of backup and archival are for an embedded system. Consumers do not back up their telephones or refrigerators, yet they do (or should) back up their personal computers or personal digital assistants. For the remainder of this paper, we assume that some form of backups are required for gizmo databases (consider manually re-programming a telephone that includes functionality to compare and select long-distance services based on the caller's calling pattern and connection times). Furthermore, these backups must be completely transparent and must not interrupt service, as users should not be aware that their gizmos are being backed up, will not remember to initiate the backups explicitly, and are unwilling to wait for service.

Data compaction or reorganization has traditionally required periodic dumping and restoration of database tables and the recreation of indices in order to bound lookup times and minimize database growth. In an embedded system, compaction and reorganization must happen automatically.

Recovery issues are similar in embedded and traditional environments with two significant exceptions. While a few seconds or even a minute of recovery is acceptable for a large server installation, consumers are unwilling to wait for an appliance to reboot. As with archival, recovery must be nearly instantaneous in an embedded product. Additionally, it is often the case that a system will be completely reinitialized, rather than performing any type of recovery, especially in systems that do not incorporate non-volatile memory. In this case, the embedded database must be restored to its initial state and must not leak any resources. This is not typically an issue for large, traditional database servers.

2.2 The Developer Perspective

In addition to the maintenance-free operation required of embedded database systems, there are a number of requirements based on the constrained resources typically available to the "gizmos" using gizmo databases. These requirements are a small disk and memory footprint, short code-path, programmatic interface (for tight application coupling and to avoid the time and size overhead of interfaces such as SQL and ODBC), support for entirely memory-resident operation (e.g., systems where file systems are unavailable), application configurability and flexibility, and support for multi-threading.

A small footprint and short code-path are self-explanatory; however, what is not as obvious is that the programmatic interface requirement is their logical result. Traditional interfaces, such as ODBC and SQL, add a significant size overhead and frequently add multiple context/thread switches per operation, not to mention several IPC calls. An embedded product is unlikely to require the complex and powerful query processing that SQL enables. Instead, in the embedded space, the ability for an application to obtain quickly its specific data is more important than a general query interface.

As some systems do not provide storage other than memory, it is essential that an embedded database work seamlessly in memory-only environments. Similarly, many embedded operating systems have a single address space architecture, so a fast, multi-threaded database architecture is essential for applications requiring concurrency.

In general, embedded applications run on gizmos whose native operating system support varies tremendously. For example, the embedded OS may or may not support user-level processes or multi-threading. Even if it does, a particular embedded application may or may not need it, e.g., not all applications need more than one thread of control. An embedded database must provide mechanisms to developers without deciding policy. For example, the threading model in an application is a matter of policy, and depends not on the database software, but on the hardware, operating system and library interfaces, and the application's requirements. Therefore, the data manager must provide for the use of multi-threading, but not require it, and must not itself determine what a thread of control looks like.

3 Berkeley DB: A Database for Embedded Systems

The current Berkeley DB package, as distributed by Sleepycat Software, is a descendant of the hash and B-tree access methods that were distributed with the 4BSD releases from the University of California, Berkeley. The original software (usually referred to as DB 1.85), was originally intended as a public domain replacement for the `dbm` and `ndbm` libraries that were proprietary to AT&T. Instead, it rapidly became widely used as a fast, efficient, and easy-to-use data store. It was incorporated into a number of Open Source packages including Perl, Sendmail, Kerberos, and the GNU standard C library. Versions 2.0 and later were distributed by Sleepycat Software (although they remain Open Source software) and added functionality for concurrency, logging, transactions, and recovery.

Berkeley DB is the result of implementing database functionality using the UNIX tool-based philosophy. Each piece of base functionality is implemented as an independent module, which means that the subsystems can be used outside the context of Berkeley DB. For example, the locking subsystem can be used to implement general-purpose locking for a non-DB application and the shared memory buffer pool can be used for any application performing page-based file caching in main memory. This modular design allows application designers to select only the functionality necessary for their application, minimizing memory footprint and maximizing performance. This directly addresses the small footprint and short code-path criteria mentioned in the previous section.

As Berkeley DB grew out of a replacement for `dbm`, its primary implementation language has always been C and its interface has been programmatic. The C interface is the native interface, unlike many database systems where the programmatic API is a layer on top of an already-costly query interface (e.g. embedded SQL). Berkeley DB's heritage is also apparent in its data model; it has none. The database stores unstructured key/data pairs, specified as variable length byte strings. This leaves schema design and representation issues the responsibility of the application, which is ideal for an embedded environment. Applications retain full control over specification of their data types, representation, index values, and index relationships. In other words, Berkeley DB provides a robust, high-performance, keyed storage system, not a particular database management system. We have designed for simplicity and performance, trading off the complex, general purpose support found in historic data management systems. While that support is powerful and useful in many applications, the overhead imposed (regardless of its usefulness to any single application), is unacceptable in embedded applications.

Another element of Berkeley DB's programmatic interface is its customizability. Applications can specify Btree comparison and prefix compression functions, hash functions, error routines, and recovery models. This means that an embedded application is able to tailor the underlying database to best suit its data demands and programming model. Similarly, the utilities traditionally bundled with a database manager (e.g., recovery, dump/restore, archive) are implemented as tiny wrapper programs around library routines. This means that it is not necessary to run separate applications for the utilities. Instead, independent threads can act as utility daemons, or regular query threads can perform utility functions. Many of the current products built on Berkeley DB are bundled as a single large server with independent threads that perform functions such as checkpoint, deadlock detection, and performance monitoring.

As described earlier, living in an embedded environment requires flexible management of storage. Berkeley DB does not require any preallocation of disk space for log or data files. While typical commercial database systems take complete control of a raw disk device, Berkeley DB cooperates with the native system's file system, and can therefore safely and easily share the file system with other applications. All databases and log files are native files of the host environment, so whatever standard utilities are provided by the environment (e.g., UNIX `cp`) can be used to manage database files as well.

Berkeley DB provides three different memory models for its management of shared information. Applications can use the IEEE Std 1003.1b-1993 (POSIX) `mmap` interface to share data, they can use system shared memory, as frequently provided by the `shmget` family of interfaces, or they can use per-process heap memory (e.g., `malloc`). Applications that require no permanent storage on systems that do not provide shared memory facilities can still use Berkeley DB by requesting strictly private memory and specifying that all databases be memory-resident. This provides pure-memory operation.

Finally, Berkeley DB is designed for rapid start-up, e.g., recovery happens automatically as part of system initialization. This means that Berkeley DB works correctly in environments where gizmos are shut down without warning and restarted.

4 Extensions for Embedded Environments

All the features described in the previous section are useful both in conventional systems as well as in embedded environments. In this section, we discuss a number of features and "automatic knobs" that are specifically geared toward the more constrained environments found in gizmo databases.

4.1 Automatic compression

Following the programmatic interface design philosophy, we support application-specific (or default) compression routines. These can be geared toward the particular data types present in the application's dataset, thus providing better compression than a general-purpose routine. Applications can specify encryption functions as well, and create encrypted databases instead of compressed ones. Alternately, the application might specify a function that performs both compression and encryption.

As applications are also permitted to specify comparison and hash functions, an application can chose to organize its data based either on uncompressed and clear-text data or compressed and encrypted data. If the application indicates that data should be compared in its processed form (i.e., compressed and encrypted), then the compression and encryption are performed on individual data items and the in-memory representation retains these characteristics. However, if the application indicates that data should be compared in its original form, then entire pages are transformed upon being read into or written out of the main memory buffer cache. These two alternatives provide the flexibility to choose the correct relationship between CPU cycles, memory and disk space and security.

4.2 In-memory logging & transactions

One of the four key properties of transaction systems is durability. This means that transaction systems are designed for permanent storage (most commonly disk). However, as described above, embedded systems do not necessarily contain any such storage. Nevertheless, transactions can be useful in this environment to preserve the semantic integrity of the underlying storage. Berkeley DB optionally provides logging functionality and transaction support regardless of whether the database and logs are on disk or in memory.

4.3 Remote Logs

While we do not expect users to backup their television sets and toasters, it is certainly reasonable that a set-top box provided by a cable carrier will need to be backed up by the cable carrier. The ability to store logs remotely can provide "information appliance" functionality, and can also be used in conjunction with local logs to enhance reliability. Furthermore, remote logs provide for true catastrophic recovery, e.g., loss of the gizmo, destruction of the gizmo, etc.

4.4 Application References to Database Buffers

In typical database systems, data must be copied from the data manager's buffer cache (or data pages) into the application's memory when it is returned to the application. However, in an embedded environment, the robustness of the total software package is of paramount importance, and maintaining an artificial isolation between the application and the data manager offers little advantage. As a result, it is possible for the data manager to avoid copies by giving applications direct references to data items in the shared memory cache. This is a significant performance optimization that can be allowed when the application and data manager are tightly integrated.

4.5 Recoverable database creation/deletion

In a conventional database management system, the creation of database tables (relations) and indices are heavyweight operations that are not recoverable. This is not acceptable in a complex embedded environment where instantaneous recovery and robust operation in the face of all types of database operations are essential. Berkeley DB provides transaction-protected file system utilities that allow us to recover both database creation and deletion.

4.6 Adaptive concurrency control

The Berkeley DB package uses page-level locking by default. This trades off fine-grained concurrency control for simplicity during recovery. (Finer-grained concurrency control can be obtained by reducing the page size in the database or the maximum number of items permitted on each page.) However, when multiple threads of control perform page-locking in the presence of writing operations, there is the potential for deadlock. As some environments do not need or desire the overhead of logging and transactions, it is important to provide the ability for concurrent access without the potential for deadlock.

Berkeley DB provides an option to perform coarser grained, deadlock-free locking. Rather than locking on pages, locking is performed at the interface to the database. Multiple readers or a single writer are allowed

to be active in the database at any instant in time, with conflicting requests queued automatically. The presence of cursors, through which applications can both read and write data, complicates this design. If a cursor is currently being used for reading, but will later be used to write, the system will be deadlock-prone unless special precautions are taken. To handle this situation, the application must specify any future intention to write when a cursor is created. If there is an intention to write, the cursor is granted an intention-to-write lock, which does not conflict with readers, but does conflict with other intention-to-write and write locks. The end result is that the application is limited to a single potentially writing cursor accessing the database at any point in time.

Under periods of low contention (but potentially high throughput), the normal page-level locking provides the best overall throughput. However, as contention rises, so does the potential for deadlock. At some cross-over point, switching to the less concurrent, but deadlock-free locking protocol will ultimately result in higher throughput as operations must never be retried. Given the operating conditions of an embedded database manager, it is useful to make this change automatically as the system detects high contention in the application's data access patterns.

4.7 Adaptive synchronization

In addition to the logical locks that protect the integrity of the database pages, Berkeley DB must synchronize access to shared memory data structures, such as the lock table, in-memory buffer pool, and in-memory log buffer. Each independent module uses a single mutex to protect its shared data structures, under the assumption that operations that require the mutex are very short and the potential for conflict is low. Unfortunately, in highly concurrent environments with multiple processors present, this assumption is not always true. When this assumption becomes invalid (that is, we observe significant contention for the subsystem mutexes), Berkeley DB can switch over to a finer-grained concurrency model for the mutexes. Once again, there is a performance trade-off. Fine-grained mutexes impose a penalty of approximately 25% (due to the increased number of mutex acquisitions and releases for each operation), but allow for higher overall throughput. Using fine-grained mutexes under low contention would cause a decrease in performance, so it is important to monitor the system carefully, so that the change happens only when it will increase system throughput without jeopardizing latency.

4.8 Source code availability

Finally, full source code is provided for Berkeley DB. This is an absolute requirement for debugging embedded applications. As the library and the application share an address space, an error in the application can easily manifest itself as an error in the data manager. Without complete source code for the library, debugging easy problems can be difficult, and debugging difficult problems can be impossible.

5 Footprint of an Embedded System

Embedded database systems compete on disk and memory footprint as well as the traditional metrics of features, price and performance. In this section of the paper, we focus on footprint.

Oracle reports that Oracle Lite 3.0 requires 350 KB to 750 KB of memory and approximately 2.5 MB of hard disk space [7]. This includes drivers for interfaces such as ODBC and JDBC. In contrast, Berkeley DB ranges in size from 75 KB to under 200 KB, foregoing heavyweight interfaces such as ODBC and JDBC. At the low end, applications requiring a simple single-user access method can choose from either extended linear hashing, B+ trees, or record-number based retrieval and pay only the 75 KB space requirement. Applications requiring all three access methods will observe the 110 KB footprint. At the high end, a fully recoverable, high-performance system occupies less than a quarter megabyte of memory. This is a system you can easily incorporate in a toaster oven. Table 1 shows the per-module breakdown of the Berkeley DB library. (Note, however, that this does not include memory used to cache database pages.)

6 Related Work

Every three to five years, leading researchers in the database community convene to identify future directions in database research. They produce a report of this meeting, named for the year and location of the meeting. The most recent of these reports, the 1998 Asilomar report, identifies the embedded database market as one of the high growth areas in database research [1]. Not surprisingly, market analysts identify the embedded database market as a high-growth area in the commercial sector as well [5].

The Asilomar report identifies a new class of database applications, which they term "gizmo" databases, small databases embedded in tiny mobile appliances, e.g., smart-cards, telephones, personal digital assistants. Such databases must be self-managing, secure and reliable. They will require plug

Subsystem	Object Size (in bytes)		
	Text	Data	Bss
Btree-specific routines	28812	0	0
Recno-specific routines	7211	0	0
Hash-specific routines	23742	0	0
Memory Pool	14535	0	0
Access method common code	23252	0	0
OS compatibility	4980	52	0
Support utilities	6165	0	0
Full Btree	77744	52	0
Full Recno	84955	52	0
Full Hash	72674	52	0
All Access Methods	108697	52	0
Locking	12533	0	0
Recovery	26948	8	4
Logging	37367	0	0
Full Package	**185545**	**60**	**4**

Table 1. Berkeley DB memory footprint.

and play data management with no database administrator (DBA), no human configurable parameters, and the ability to adapt to changing conditions. More specifically, the Asilomar authors claim that the goal is self-tuning, including defining the physical and logical database designs, generating reports, and executing utilities [1]. To date, few researchers have accepted this challenge, and there is a dearth of research literature on the subject.

Our approach to embedded database administration is fundamentally different from that described by the Asilomar authors. We adopt their terminology, but view the challenge in supporting gizmo databases to be that of self-sustenance *after* initial deployment. We find it, not only acceptable, but desirable to expect application developers to control initial database design and configuration. To the best of our knowledge, none of the published work in this area addresses this approach.

As the research community has not provided guidance in this arena, most work in embedded database administration has been done by the commercial vendors. These vendors can be partitioned into two groups: companies selling databases specifically designed for embedding or programmatic access and the major database vendors (e.g., Oracle, Informix, Sybase). The embedded vendors all acknowledge the need for automatic administration, but normally fail to identify precisely how their products accomplish this. A notable exception is Interbase, whose white paper comparison with Sybase and Microsoft's SQL servers explicitly addresses features of maintenance ease. Interbase states that as they use no log files, there is no need for log reclamation, checkpoint tuning, or other tasks normally associated with log management. However, Interbase uses Transaction Information Pages, and it is unclear how these are reused or reclaimed [6]. Additionally, with a log-free system, they must use a FORCE policy (flushing all modified pages to disk at commit), as defined by Haerder and Reuter [4]. This has serious performance consequences for disk-based systems. Berkeley DB does use logs and therefore requires log reclamation, but provides interfaces such that applications may reclaim logs safely and programmatically. While Berkeley DB requires checkpoints, the goal of tuning the checkpoint interval is to bound recovery time. The checkpoint interval can be expressed as an amount of log data written. The application designer sets a target recovery time, and selects the amount of log data that can be read in that interval and specifies the checkpoint interval appropriately. Even as load changes, the time to recover does not.

The backup approaches taken by Interbase and Berkeley DB are similar in that they both allow online backup, but rather different in their effect on transactions running during backup. As Interbase performs backups as transactions [6], concurrent queries potentially suffer long delays. Berkeley DB uses native operating system utilities and recovery for backups, so there is no interference with concurrent activity, other than potential contention on disk arms.

There are a number of other vendors selling products into the embedded market (e.g., Raima, Centura, Pervasive, Faircom), but none of these highlight the special requirements of embedded database applications. On the other end of the spectrum, the major vendors (e.g., Oracle, Sybase, Microsoft), are all clearly aware of the importance of the embedded market. As discussed earlier, Oracle has announced its Oracle Lite server for embedded use. Sybase has announced its UltraLite platform for "application-optimized, high-performance, SQL database engine for professional application developers building solutions for mobile and embedded platforms" [8]. We believe that SQL is fundamentally incompatible with the gizmo database environment and the truly embedded systems for which Berkeley DB is most suitable. Microsoft research is taking a different approach, developing technology to assist in automating initial database design and index specification [2][3]. As described earlier, such configuration is, not only acceptable in the embedded market, but desirable so that application designers can tune their entire applications for the target environment.

7 Conclusions

The coming wave of embedded systems poses a new set of challenges for data management. The traditional server-based, large footprint systems designed for high performance on big iron are not the correct technical answer for this environment. Instead, application developers need small, fast, versatile data management tools that can be integrated easily within specific application environments. In this paper, we have identified several of the key issues in designing these systems and shown how Berkeley DB provides many of these requirements.

8 References

[1] Bernstein, P., Brodie, M., Ceri, S., DeWitt, D., Franklin, M., Garcia-Molina, H., Gray, J., Held, J., Hellerstein, J., Jagadish, H., Lesk, M., Maier, D., Naughton, J., Pirahesh, H., Stonebraker, M., Ullman, J., "The Asilomar Report on Database Research," *SIGMOD Record 27*(4): 74–80, 1998.

[2] Chaudhuri, S., Narasayya, V., "AutoAdmin 'What-If' Index Analysis Utility," *Proceedings of the ACM SIGMOD Conference*, Seattle, 1998.

[3] Chaudhuri, S., Narasayya, V., "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server," *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997.

[4] Haerder, T., Reuter, A., "Principles of Transaction-Oriented Database Recovery," *Computing Surveys 15*,4 (1983), 237–318.

[5] Hostetler, M., "Cover Is Off A New Type of Database," Embedded DB News, `http://www.theadvisors.com/embeddeddb-news.htm`, 5/6/98.

[6] Interbase, "A Comparison of Borland InterBase 4.0 Sybase SQL Server and Microsoft SQL Server," `http://web.interbase.com/products/doc_info_f.html`.

[7] Oracle, "Oracle Delivers New Server, Application Suite to Power the Web for Mission-Critical Business," `http://www.oracle.com.sg/partners/news/newserver.htm`, May 1998.

[8] Sybase, Sybase UltraLite, `http://www.sybase.com/products/ultralite/beta`.

[9] Transaction Processing Council, "TPC-C Benchmark Specification, Version 3.4," San Jose, CA, August 1998.

[10] Transaction Processing Council, "TPC-D Benchmark Specification, Version 2.1," San Jose, CA, April 1999.

Embedded Systems Digest

Peter H. Salus, Rapporteur

Though most folks aren't conscious of it, their homes and workplaces are full of computers—or at least parts of computers. Cars, bread machines, TVs, VCRs, coffeemakers, rice cookers, dish and clothes washers, etc., etc., are full of computing power.

The Workshop on Embedded Systems, co-sponsored by the MIT Media Lab and USENIX, spent three days considering the issues and the problems.

The wide success of Hasbro's "Furby" has demonstrated just how much we can do with a six-for-a-dollar chip.

Among the "fun" things considered were "things that think": intelligent refrigerators, pantries, and cutting boards, where food freshness and suchlike would be continually monitored. ("Your can of tomatoes should be used or thrown away within 10 days.") Just what happens to someone like me, who buys yogurt and keeps it weeks beyond its expiry, so that it separates and I pour off the whey to cook with the residue, is unclear.

One extremely interesting talk was by David Smith, president and CEO of Object Automation, Inc. "The promise of object-oriented technology has long been that of reusable, portable software components," he said. "Contemporary computing technologies, like COM and CORBA, coupled with those that accompany the evolution of personal computing . . . the PC itself, plus networking and internetworking, have provided an environment where this promise has become a reality. The benefits that derive from the reality of truly reusable software components are enormous, even though much of that benefit is yet to be realized." He emphasized factory improvements, as they spur the "fourth industrial revolution."

Talking to your box of breakfast cereal may not be in vain in the future. Whether it will be rewarding is another question. I had a vision of foodstuffs and other household products clamoring at me as I freewheel down the aisles of my supermarket. No, thanks.

Some of the graduate students at the Lab are experimenting with building circuits on paper with a Xerox machine. Metallic inks seem to work fine and (experimentally) they can do away with the chip in packaging. (They did a box of mix that responds to pressure on a list with the print of that specific recipe.) Perhaps there will be an electronically dense wrap so that we can cut off the packages' eyes and ears.

But the creation of neat things isn't all.

Session: Computer Everywhere

The Personal Node (PN)

Gregory G. Finn and Joe Touch, University of Southern California

PN is a small wallet-card that connects to the Internet. Among the points made were:

- Bodies on line: heart attacks not treated in time cost $130 billion a year.
- Remote needs: objects that can find us.

The most interesting questions and answers:

- How does one distinguish what's important? Simple set of buttons; half-size of PalmPilot; then shrink down.
- Geographic addressing? Perhaps GPS.
- Authentication for financial transactions? We'd like this—mapped to user.
- Policy issues: privacy; tracking. What does privacy mean? We don't know. What about anonymity?
- How does the device know good guys from bad guys? There are more than merely aggregation problems.

Several folks questioned the value of IR and RF links and one remarked that the authors were wrong on the IP issue.

Discourse with Disposable Computers: How and Why You Will Talk to Your Tomatoes

David Arnold, Bill Segall, Julian Boot, Andy Bond, Melfyn Lloyd, and Simon Kaplan, Distributed Systems Technology Centre

These guys have read too much Neal Stephenson. Picture Hiro's pizza box extrapolated to chewing-gum wrappers. Now, anticipate the clamor as you pass a trash can. Think of the event volume.

The most pressing discussion concerned the "noise" level; the notion of "content-based routing"; and the fact that a subscription model is dependent on receivers, not senders.

Smart Office Spaces
Bora A. Akyol, Matt Fredette, Alden W. Jackson, Rajesh Krishnan, David Mankins, Craig Partridge, Nicholas Shectman, and Gregory D. Troxel, BBN Technologies

This was an entertaining, "toy" paper. I was irritated by some sexist remarks ("a confident touch typist should be able to edit a file if she has access to a keyboard . . . ") in the paper, but they're not "content." It is over 30 years since I was introduced to the notion of the paperless office. I'm not holding my breath. However, the concept of "smart spaces" is fetching in a sci-fi movie, if not at large.

Scaling was the most important topic of discussion. Working within one office, or even a group of offices, lacks the real complexities of the world. Moreover, with a sea of devices, how much security can there be? Will we end up with "stealth management"?

Session: Networking Infrastructure

***AirJava:* Networking for Smart Spaces**
Kevin L. Mills, National Institute of Standards and Technology

This was a different presentation on smart spaces. Mills sees us as "wireless islands" in a "global wired ocean." He foresees "many kinds of multicasting."

He was queried about "active construction" of a combination between Linux and JavaVM. Mills said they were "beginning construction" in the summer.

Further questions involved:

- Widget replication—we need core functionality, not JavaVM that gets in the way when you start playing with Jini.
- The financial threshold: about $10.
- The NRC panel on power for the "dismounted soldier." We need new and better battery technology.

Finally, it was noted that none of the attendees was a "hardware guy," and that those are the folks who may be most important.

Bringing the Internet to All Electronic Devices
Michael Howard and Christopher S. Sontag, emWare, Inc.

This paper began with an assumption that I thought was important and well-articulated: What's the ultimate value of networking all these devices? We must make certain that the cost involved is lower than the gain. Considerations include processor performance, power, memory, storage, and network limitations. Understanding costs and the problems involving end-user education are very important.

Questions that struck me:

- How do we change program locations? We reload portions or whole applications.
- How do we program? What sort of APIs can we have? We definitely need very flexible products.
- Security? With so many devices isn't physical security a problem, too? There's much work to be done.

RETHER: A Software-Only Real-Time Ethernet for PLC Networks
Tzi-cker Chiueh, State University of New York at Stony Brook

RETHER is a software-only realtime Ethernet for networks made up of programmable logic controllers. I really liked this paper because it constrained its domain: PLCs are used overwhelmingly in industrial automation. At the same time, the software involved must be both fault-tolerant and not socket-based.

The discussion was quite animated. The use of single shared media assumes a point-to-point link, and whether each node needs to know which other node is using RT was among the topics. The speed of bandwidth growth was another subject. This led to the question of the capacity/performance curve and whether service guarantees were feasible. The final queries dealt with statistical loading and efficiency.

Session: Design

Pebble, A Component-Based Operating System for Embedded Applications
John Bruno, José Brustoloni, Eran Gabber, Avi Silberschatz, and Christopher Small, Lucent Technologies—Bell Laboratories

Pebble is an OS designed for high-end embedded communications devices. It comprises a set of replaceable, user-level components. There are benchmark results that appear to demonstrate Pebble's effectiveness.

- What about priorities? Done inside the scheduler—interrupts are converted to schedules and delivered immediately, but not acted upon immediately.
- Portal generation is a one-time event.

- Is portal generation interruptable? Only when you are placing into the table.
- Untrusted components: you can still spoof the system. Software checking is needed.
- The scheduler knows about all the threads: "If you can't trust your scheduler, who can you trust?"

Massively Distributed Systems: Design Issues and Challenges
Dan Nessett, 3Com Corporation

Nessett envisions systems involving billions of nodes; the paper discusses the engineering problems this entails. The discussion included questions about unifying principles, measurement, testing and debugging, etc.

- What about the health industry? It's never an early adopter.
- Is there a way to control complexity through software? Central control is an impossibility.

Session: Control

Learning in Intelligent Embedded Systems
Daniel D. Lee, Lucent Technologies—Bell Laboratories, and H. Sebastian Seung, Massachusetts Institute of Technology

Lee and Seung have built a robot dog that learns to identify sounds and faces. It looks great, does its thing, and is built out of $700 worth of off-the-shelf parts. The mathematics was worth the price of admission. Watching the "dog" "walk" and search and identify was very impressive. It's the perfect pet for a small apartment.

Using Mobile Code Interfaces to Control Ubiquitous Embedded Systems
Kari Kangas and Juha Röning, University of Oulu

All the code appeared to be in Java; insecurity is still a major drawback. The discussion focused on:

- different kinds of mobile-code protocols
- comparisons with Inferno involving the fact that this avoids using only one language. (37KB of code)
- uses of code standardization

Challenges in Embedded Database System Administration
Margo I. Seltzer, Harvard University, and Michael A. Olsen, Sleepycat Software

This was a really interesting paper because of the topics covered and the points made.

While the paper per se involved database architecture and the problems entailed by administration of embedded databases, perhaps the best portion of the paper was its list of aphorisms, some of which were:

1. SQL sucks.
2. Size matters.
3. Speed matters.
4. You don't need the whole Swiss army knife.
5. Service, service, service.
6. You get paid a lot, your life should difficult; but the software should be bulletproof when it's rolled out.
7. Underestimate your user.
8. People hate surprises.
9. Resilience in the face of failures is important.

I'll stop here. The discussion involved the sheer size of some of the code; that complexity is still something we don't have a handle on; and that by-and-large humans adapt to failure.

Discussion

What Have We Learned? Where Do We Go?
Dan Geer, CertCo

Workshop co-chair Dan Geer led a (to me) interesting and important discussion concerning a variety of points central to the notion of the future of workshops like this one.

One participant, for example, said that he felt the workshop had been too diffuse, "all over the place," and (though interesting) had contained nothing he found of value. There were so many pieces—networking, security, ubiquity, control, robustness, etc., etc.—that he was like the blind men and the elephant.

Geer asked, "Who or what is missing?" Hardware guys, chip designers, etc., appeared to be the most obvious. The DNS folks and the IPv6 folks were also absent.

It was obvious that while no one was willing to define what an embedded system is, everyone thought they'd know one when they saw one. Even the question of what exactly a device is was brought up.

After a discussion of resilience and recovery, "Do we standardize?" was asked. Co-chair Mike Hawley indicated a strong "no." "Standards are an Ice Age," he said.

It became clear that we should become accustomed to living with incoherence (identity, language, scope of detection); that important questions were: "how do I talk to you?" and "explain yourself."

There was a lot of talk about hybrid vigor. Diversity is important in development. This vigor needs to be felt in domestic technology, factories, transportation, and office services. If it works (well enough), it will get installed.

The workshop concluded with a proposal that it be done "again, only better." Make it three days once a year or two days twice a year, preferably the latter. Keep the joint Media Lab / USENIX / Cambridge venue until we have a reason not to. Each morning is a case study of an actual deployed system—invited on the basis of its known relevance and using demos if at all practical. Make the afternoon some mix of theoretical results and general argument, possibly broken out into topic groups. Keep the number of participants small but get reps from the areas that should have been there this time but were not. Keep the workshop style as long as possible. Have, if someone will do it, some organized sense of what is unsolved (such as whether every device should be required to "explain itself" on demand) and keep a scorecard on progress. Pay as much attention to what has failed in the market (of ideas or of money) and why. Keep a place for refereed work (so those with a need for formal bibliography growth can participate) but have no qualms about inviting the rest of the event. Continue to have meals together but arrange both meeting and eating spaces to maximize conversation. Evolve into side-by-side events if and only if specialization is essential to progress. Geer: "Think big—somebody will, and why not us?"

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

SAGE, the System Administrators Guild

The System Administrators Guild, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

Member Benefits:

- Free subscription to *;login:*, the Association's magazine, published eight–ten times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and C++, book and software reviews, summaries of sessions at USENIX conferences, and Snitch Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, via the USENIX Online Library on the World Wide Web.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as object-oriented technologies, security, operating systems, electronic commerce, and NT – as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.
- Discount on BSDI, Inc. products.
- Discount on all publications and software from Prime Time Freeware.
- Savings (10-20%) on selected titles from Academic Press, Morgan Kaufmann, New Riders/Cisco Press/MTP, O'Reilly & Associates, OnWord Press, The Open Group, Sage Science Press, and Wiley Computer Publishing.
- Special subscription rates for Cutter Consortium newsletters, *The Linux Journal, The Perl Journal, IEEE Concurrency, Server/Workstation Expert, Sys Admin Magazine*, and all Sage Science Press journals.

Supporting Members of the USENIX Association:

C/C++ Users Journal
Cirrus Technologies
Cisco Systems, Inc.
CyberSource Corporation
Deer Run Associates
Greenberg News Networks/MedCast Networks
Hewlett-Packard India Software Operations
Internet Security Systems, Inc.
JSB Software Technologies
Microsoft Research
MKS, Inc.
Motorola Australia Software Centre
NeoSoft, Inc.
New Riders Press
Nimrod AS
O'Reilly & Associates Inc.
Performance Computing
Questra Consulting
Sendmail, Inc.
Server/Workstation Expert
TeamQuest Corporation
UUNET Technologies, Inc.
Windows NT Systems Magazine
WITSEC, Inc.

Sage Supporting Members:

Atlantic Systems Group
Collective Technologies
D. E. Shaw & Co.
Deer Run Associates
Electric Lightwave, Inc.
ESM Services, Inc.
GNAC, Inc.
Mentor Graphics Corp.
Microsoft Research
MindSource Software Engineers
Motorola Australia Software Centre
New Riders Press
O'Reilly & Associates Inc.
Remedy Corporation
RIPE NCC
SysAdmin Magazine
Taos Mountain
TransQuest Technologies, Inc.
Unix Guru Universe

For further information about membership, conferences or publications, contact:
USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA.
Phone: 510-528-8649. Fax: 510-548-5738.
Email: *office@usenix.org.*
URL: *http://www.usenix.org.*

ISBN 1-880446-31-6